Experiment 2

Student Name: Nitil Jakhar **UID:** 22BCS17300
Branch: CSE **Section/Group:** NTPP_IOT-602/A
Semester: 6th **Date of Performance:** 3/02/2
Subject: AP 2

1. **Aim: Linked Lists**
2. **Objective:**
    1. Remove duplicates from a sorted list
    2. Detect a cycle in a linked list
3. **Code:**

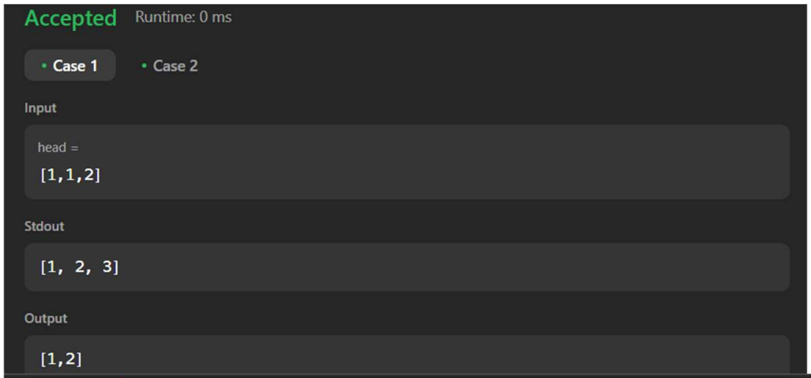
1. Remove duplicates from a sorted list:

```
from typing import Optional

class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

class Solution:
    def deleteDuplicates(self, head: Optional[ListNode]) -> Optional[ListNode]:
        current = head

        while current and current.next:
            if current.val == current.next.val:
                current.next = current.next.next  # Skip duplicate node
            else:
                current = current.next  # Move to the next unique node

        return head

# Helper function to convert a list to a linked list
def list_to_linked_list(arr):
    if not arr:
        return None
    head = ListNode(arr[0])
    current = head
    for val in arr[1:]:
        current.next = ListNode(val)
        current = current.next
    return head

# Helper function to convert a linked list to a list
def linked_list_to_list(head):
    result = []
    current = head
    while current:
        result.append(current.val)
        current = current.next
    return result

# Example usage:
head = list_to_linked_list([1, 1, 2, 3, 3])
solution = Solution()
new_head = solution.deleteDuplicates(head)
print(linked_list_to_list(new_head))  # Output: [1, 2, 3]
```

2. **Detect a cycle in a linked list:**

```
from typing import Optional

class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

class Solution:
    def hasCycle(self, head: Optional[ListNode]) -> bool:
        slow = head  # Slow pointer moves one step at a time
        fast = head  # Fast pointer moves two steps at a time

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

            if slow == fast:  # If slow meets fast, a cycle exists
                return True
        return False  # If the loop exits, no cycle is present

# Helper function to create a linked list with a cycle
def create_linked_list_with_cycle(arr, pos):
    if not arr:
        return None

    head = ListNode(arr[0])
    current = head
    cycle_node = None

    for i in range(1, len(arr)):
        current.next = ListNode(arr[i])
        current = current.next
        if i == pos:
            cycle_node = current  # Store reference to the cycle node

    if pos != -1:
        current.next = cycle_node  # Create cycle

    return head

# Example usage:
head = create_linked_list_with_cycle([3, 2, 0, -4], 1)
solution = Solution()
print(solution.hasCycle(head))  # Output: True
```

4. **Output:**
   1. **Remove duplicates from a sorted list:**



Input

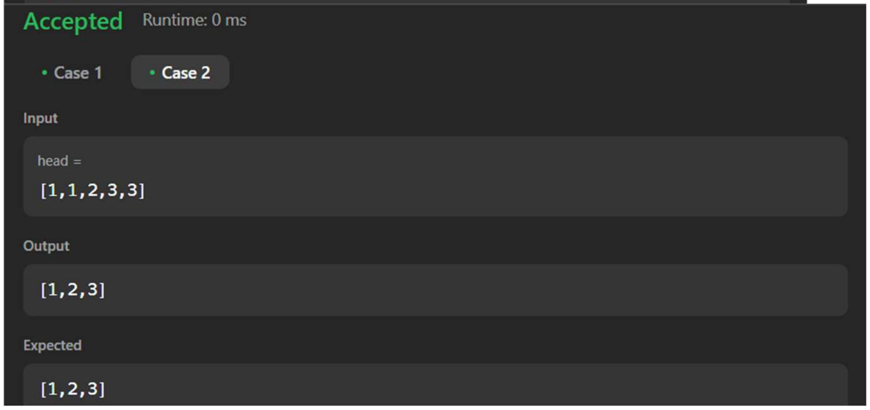
```
head =
[1,1,2]
```

Stdout

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

Output

```
[1,2]
```



Input

```
head =
[1,1,2,3,3]
```

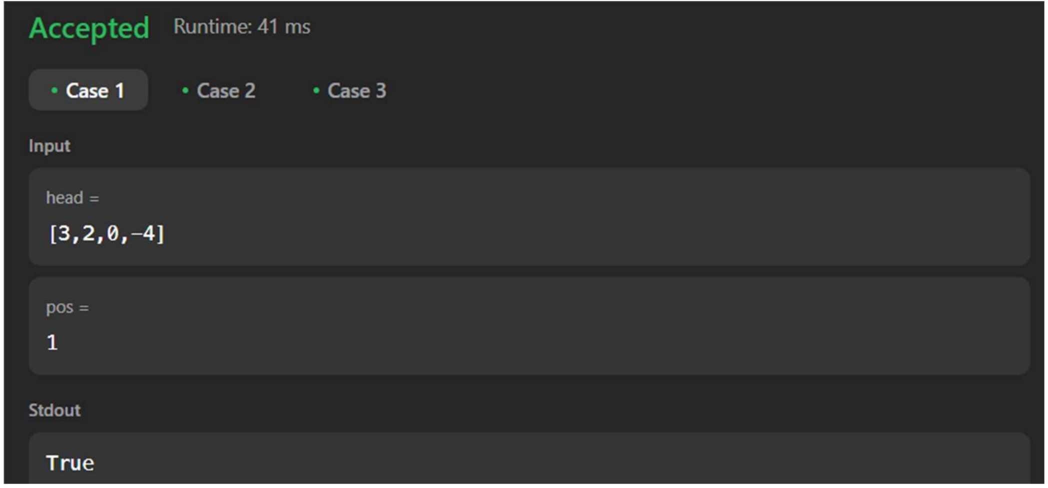
Output

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

Expected

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

2. **Detect a cycle in a linked list:**



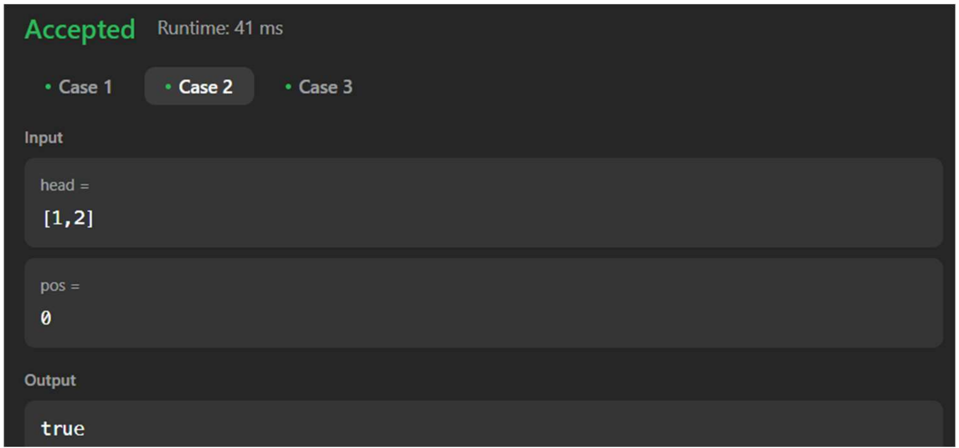
Input

```
head =
[3,2,0,-4]
```

```
pos =
1
```

Stdout

```
True
```



Accepted Runtime: 41 ms

• Case 1 • Case 2 • Case 3

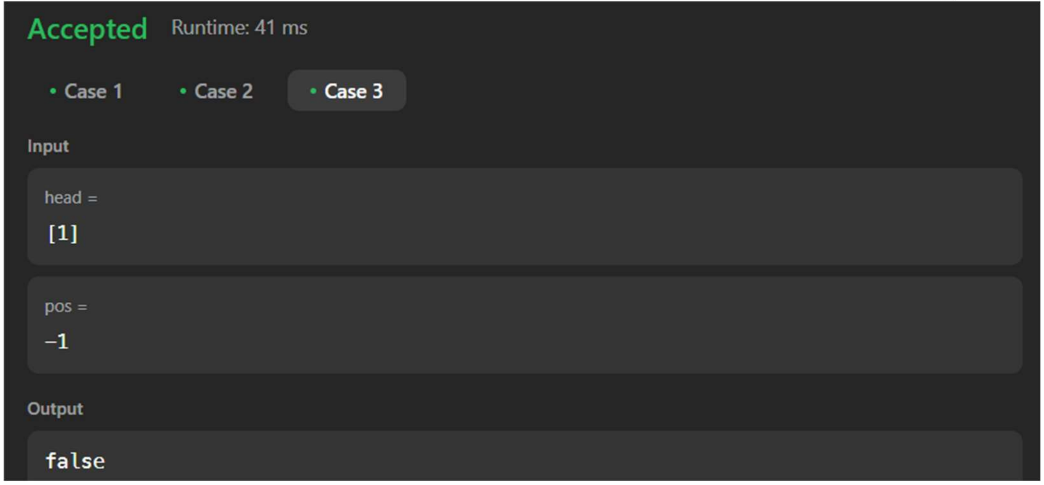
Input

head =
[1,2]

pos =
0

Output

true



Accepted Runtime: 41 ms

• Case 1 • Case 2 • Case 3

Input

head =
[1]

pos =
-1

Output

false

5. Learning Outcome

1) Learned how to traverse and modify a **sorted** linked list by removing duplicate elements while maintaining the original order.
2) Gained insight into how to remove elements from a linked list without using extra space, modifying the list directly.
3) Observed how to check and skip nodes using current.next = current.next.next when duplicates are found.
4) Learned how to convert a list into a linked list and vice versa for easier testing and debugging.
5) Understood that the approach runs in **O(n) time complexity**, as each node is visited once.