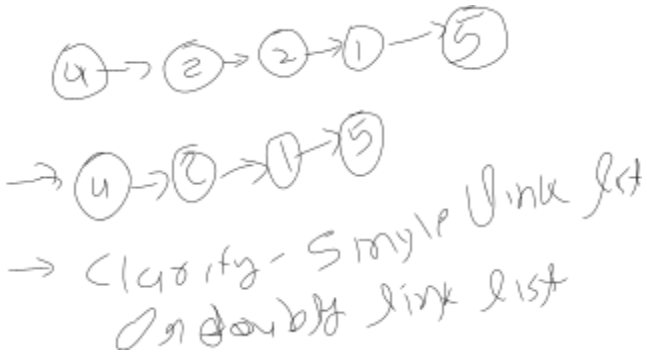
Remove Dups! Write code to remove duplicates from an unsorted linked list.
FOLLOW UP
How would you solve this problem if a temporary buffer is not allowed?
Hints: #9, #40

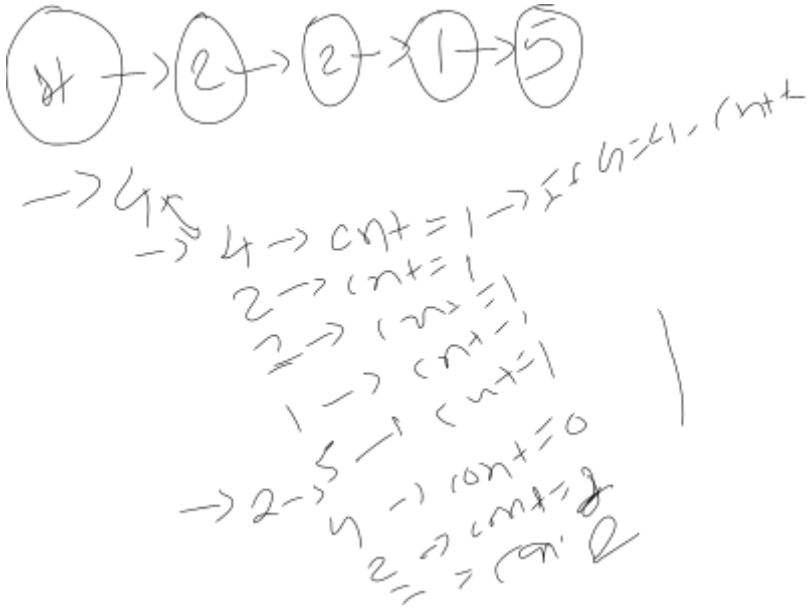



First approach: (1) Traverse through Linked list (2) Store each element in an array (3) Find out duplicates elements from an array and insert into a duplicate list array (4) Create a method to delete resultant set from a linked list (5) return the liked list

Second Approach: (1) Traverse through liked lists and add each element in Hashset where it will automatically remove the duplicate. Define a new linked list and add each element from hash set and replace the linked list with the newly created linked list. --> Will develop the second approach
Suppose we have LinkedListNode class

```
Class LinkedListNode{
        LinkedListNode next = null;
        int data;
        public LinkedListNode(int d){
                this.data = d;
        }
}
void deleteDup(LinkedListNode n){
        HashSet<Integer> set = new Hashset<Integer>();
        LinkedListNode previous = null;
        // iteragte through the linked list
        while(n != null){
                if(set.contains(n.data)){
                        // following we have removed the node by cutting the link
                        previous.next = n.next;
                } else {
                        set.add(n.data)
                        previous = n;
                }
                n = n.next;
        }
}
```

Follow up



21

First Approach: 2 iterators where inner iterator iterate through each node and compare with the outer iterator if matches then replace the link of the previous node with next to next node

```
void deleteDup(LinkedListNode n){
	LinkedListNode head = n;
	LinkedListNode previous = null;
	// iteragte through the linked list
	while(head != null){
		LinkedListNode fastRunner = head;
		while(fastRunner.next != null){
			if(fastRunner.next.data == head.data){
				fastRunner.next = fastRunner.next.next;
			} else{
				fastRunner = fastRunner.next;
			}
		}
		head = head.next;
	}
}
```

This code runs in O(1) space but O(n