Solution Review: Remove Duplicates from Linked List

This review provides an analysis of the solution to the Remove Duplicates from a Linked List challenge.

We'll cover the following

Solution: Using a Set/Hash Table

main.py

LinkedList.py

Node.py

```
from LinkedList import LinkedList
from Node import Node


def remove_duplicates(lst):
    current_node = lst.get_head()
    prev_node = lst.get_head()
    # To store values of nodes which we already visited
    visited_nodes = set()
    # If List is not empty and there is more than 1 element in List
    if not lst.is_empty() and current_node.next_element:
        while current_node:
            value = current_node.data
```

```
            if value in visited_nodes:
                # current_node is already in the HashS
                # connect prev_node with current_node's next element
                # to remove it
                prev_node.next_element = current_node.next_element
                current_node = current_node.next_element
                continue
            # Visiting currentNode for first time
            visited_nodes.add(current_node.data)
            prev_node = current_node
            current_node = current_node.next_element


lst = LinkedList()
lst.insert_at_head(7)
```

This is, perhaps, the most efficient way of removing duplicates from a linked list. We've seen this approach before in Challenge 10 when we detected a loop in our linked list.

Every node we traverse is added to the `visited_nodes` set. If we reach a node that already exists in the set, it must be a duplicate.

`prev_node` is used to keep track of the preceding node. This allows us to easily manipulate the previous and next nodes during the deletion of our `current_node`.

Time Complexity#

This is a linear algorithm, hence, the time complexity is *O(n)*.

Next, we'll learn how to apply **union** and **intersection** operations on linked lists.

Back

Challenge 11: Remove Duplicates fro...

Next

Challenge 12: Union & Intersection of ...

Completed

Report an Issue