

Solution Review: Remove Duplicates from Linked List

This review provides a detailed analysis of the different ways to solve the Remove Duplicates from a Linked List challenge.

We'll cover the following



- Solution: Using a Set/Hash Table
- Time Complexity







Solution: Using a Set/Hash Table

main.py

LinkedList.py

Node.py

```
1 from LinkedList import LinkedList
2 from Node import Node
3
4
5 def remove_duplicates(lst):
6     current_node = lst.get_head()
7     prev_node = lst.get_head()
8     # To store values of nodes which we already visited
9     visited_nodes = set()
10    # If List is not empty and there is more than 1 element in
11    if not lst.is_empty() and current_node.next_element:
12        while current_node:
13            value = current_node.data
14            if value in visited_nodes:
15                # current_node is already in the HashSet
16                # connect prev_node with current_node's next element
17                # to remove it
18                prev_node.next_element = current_node.next_element
19                current_node = current_node.next_element
20                continue
21            # Visiting currentNode for first time
22            visited_nodes.add(current_node.data)
23            prev_node = current_node
24            current_node = current_node.next_element
25
26
27 lst = LinkedList()
28 lst.insert_at_head(7)
29 lst.insert_at_head(7)
30 lst.insert_at_head(22)
31 lst.insert_at_head(14)
32 lst.insert_at_head(21)
33 lst.insert_at_head(14)
34 lst.insert_at_head(7)
35
36 lst.print_list()
37
38 remove_duplicates(lst)
39 lst.print_list()
40
```



Output0.182s

7 -> 14 -> 21 -> 14 -> 22 -> 7 -> 7 -> None
7 -> 14 -> 21 -> 22 -> None

This is, perhaps, the most efficient way of removing duplicates from a linked list. We've seen this approach before in Challenge 10 (<https://www.educative.io/courses/data-structures-in-python-an-interview-refresher/3w7qnv17rEQ>) 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

Next 

Challenge 11: Remove Duplicates fro...Challenge 12: Union & Intersection of ...

 Mark as Completed