

Solution Review: Remove Duplicates

This lesson contains the solution review for the challenge of removing duplicates from a doubly linked list.

WE'LL COVER THE FOLLOWING ^

- Implementation
- Explanation

In this lesson, we consider how to remove duplicates from a doubly linked list.

Implementation

Check out the code below:

```
def remove_duplicates(self):
    cur = self.head
    seen = dict()
    while cur:
        if cur.data not in seen:
            seen[cur.data] = 1
            cur = cur.next
        else:
            nxt = cur.next
            self.delete_node(cur)
            cur = nxt
```



remove_duplicates(self)

Explanation

In the method `remove_duplicates`, we will keep track of the duplicates using a Python dictionary which we declare on **line 3**. `cur` is set to `self.head` on **line 2** to help us traverse the linked list using the `while` loop on **line 4**.

In the `while` loop, we have to keep track of the number of times we encounter a data element. Therefore, if `cur.data` is not present in `seen`, we set the value of the key `cur.data` to `1` on **line 6** to indicate that we have encountered it

once while traversing the linked list. In the next line, we update `cur` to `cur.next` to iterate to the next node.

On the other hand, if `cur.data` is present in `seen`, we jump to the `else` portion on **line 8**. This implies that `cur` in the current iteration has already been encountered in the previous iterations and is present in `seen`. On **line 9**, we save the next node of `cur` in `nxt` to keep with the traversal after we remove the duplicate node. We call the class method `self.delete_node(cur)` to delete the duplicate node on **line 10** and then set `cur` to `nxt` for the next iteration on **line 11**.

Now let's discuss the `delete_node` method. You can see the modifications through the highlighted lines in the code snippet below. Instead of matching `key` with `cur.data`, we are comparing the entire `node` passed into the method with `cur`.

```
def delete_node(self, node):
    cur = self.head
    while cur:
        if cur == node and cur == self.head:
            # Case 1:
            if not cur.next:
                cur = None
                self.head = None
                return

            # Case 2:
            else:
                nxt = cur.next
                cur.next = None
                nxt.prev = None
                cur = None
                self.head = nxt
                return

        elif cur == node:
            # Case 3:
            if cur.next:
                nxt = cur.next
                prev = cur.prev
                prev.next = nxt
                nxt.prev = prev
                cur.next = None
                cur.prev = None
                cur = None
                return

            # Case 4:
            else:
                prev = cur.prev
                prev.next = None
                cur.prev = None
                cur = None
```

```
        return  
    cur = cur.next
```

```
delete_node(self, node)
```

In the code widget below, we have the entire implementation of **DoublyLinkedList** that we have learned so far in this chapter. Go ahead and explore it yourself!

```
class Node:  
    def __init__(self, data):  
        self.data = data  
        self.next = None  
        self.prev = None  
  
class DoublyLinkedList:  
    def __init__(self):  
        self.head = None  
  
    def append(self, data):  
        if self.head is None:  
            new_node = Node(data)  
            new_node.prev = None  
            self.head = new_node  
        else:  
            new_node = Node(data)  
            cur = self.head  
            while cur.next:  
                cur = cur.next  
            cur.next = new_node  
            new_node.prev = cur  
            new_node.next = None  
  
    def prepend(self, data):  
        if self.head is None:  
            new_node = Node(data)  
            new_node.prev = None  
            self.head = new_node  
        else:  
            new_node = Node(data)  
            self.head.prev = new_node  
            new_node.next = self.head  
            self.head = new_node  
            new_node.prev = None  
  
    def print_list(self):  
        cur = self.head  
        while cur:  
            print(cur.data)  
            cur = cur.next  
  
    def add_after_node(self, key, data):  
        cur = self.head  
        while cur:  
            if cur.next is None and cur.data == key:  
                self.append(data)  
                return
```

```

        elif cur.data == key:
            new_node = Node(data)
            nxt = cur.next

            cur.next = new_node
            new_node.next = nxt
            new_node.prev = cur
            nxt.prev = new_node
            return
        cur = cur.next

def add_before_node(self, key, data):
    cur = self.head
    while cur:
        if cur.prev is None and cur.data == key:
            self.prepend(data)
            return
        elif cur.data == key:
            new_node = Node(data)
            prev = cur.prev
            prev.next = new_node
            cur.prev = new_node
            new_node.next = cur
            new_node.prev = prev
            return
        cur = cur.next

def delete(self, key):
    cur = self.head
    while cur:
        if cur.data == key and cur == self.head:
            # Case 1:
            if not cur.next:
                cur = None
                self.head = None
                return

            # Case 2:
            else:
                nxt = cur.next
                cur.next = None
                nxt.prev = None
                cur = None
                self.head = nxt
                return

        elif cur.data == key:
            # Case 3:
            if cur.next:
                nxt = cur.next
                prev = cur.prev
                prev.next = nxt
                nxt.prev = prev
                cur.next = None
                cur.prev = None
                cur = None
                return

            # Case 4:
            else:
                prev = cur.prev
                prev.next = None
                cur.prev = None

```

```

        cur = None
        return
    cur = cur.next

def delete_node(self, node):
    cur = self.head
    while cur:
        if cur == node and cur == self.head:
            # Case 1:
            if not cur.next:
                cur = None
                self.head = None
                return

            # Case 2:
            else:
                nxt = cur.next
                cur.next = None
                nxt.prev = None
                cur = None
                self.head = nxt
                return

        elif cur == node:
            # Case 3:
            if cur.next:
                nxt = cur.next
                prev = cur.prev
                prev.next = nxt
                nxt.prev = prev
                cur.next = None
                cur.prev = None
                cur = None
                return

            # Case 4:
            else:
                prev = cur.prev
                prev.next = None
                cur.prev = None
                cur = None
                return
        cur = cur.next

def reverse(self):
    tmp = None
    cur = self.head
    while cur:
        tmp = cur.prev
        cur.prev = cur.next
        cur.next = tmp
        cur = cur.prev
    if tmp:
        self.head = tmp.prev

def remove_duplicates(self):
    cur = self.head
    seen = dict()
    while cur:
        if cur.data not in seen:
            seen[cur.data] = 1
            cur = cur.next

```

```
        else:
            nxt = cur.next
            self.delete_node(cur)
            cur = nxt

dllist = DoublyLinkedList()
dllist.append(8)
dllist.append(4)
dllist.append(4)
dllist.append(6)
dllist.append(4)
dllist.append(8)
dllist.append(4)
dllist.append(10)
dllist.append(12)
dllist.append(12)

dllist.remove_duplicates()
dllist.print_list()
```



Hope you had fun with this lesson! Now brace yourself for another challenge in the next lesson. All the best!