Remove Duplicates

In this lesson, we will learn how to remove duplicates from a linked list.

WE'LL COVER THE FOLLOWING

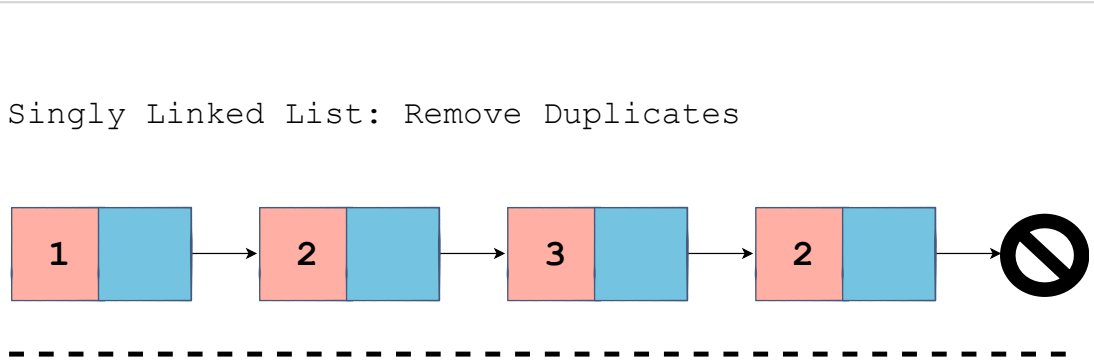
In this lesson, we will use a hash table to remove all duplicate entries from a single linked list. For instance, if our singly linked list looks like this:

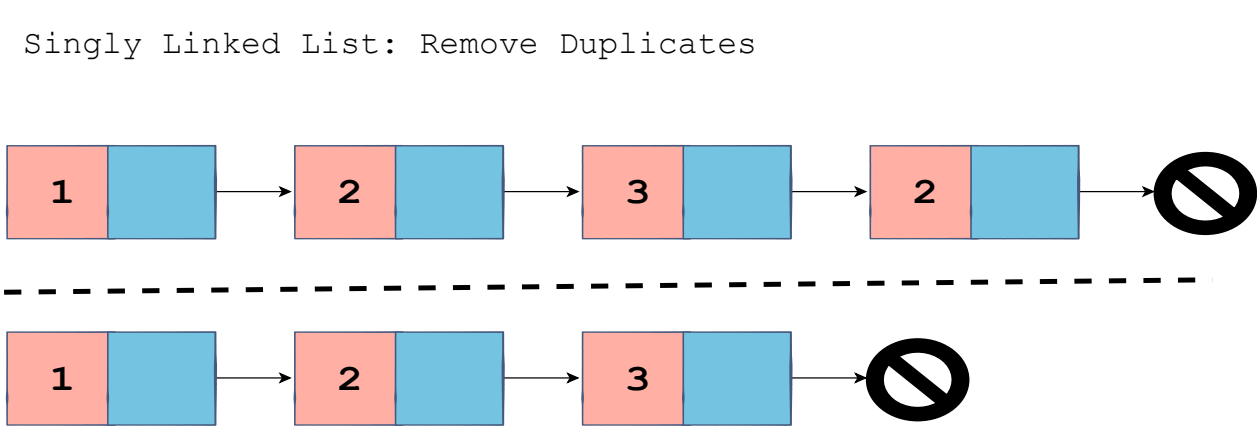
```
1 - 6 - 1 - 4 - 2 - 2 - 4
```

Then the desired resulting singly linked list should take the form:

```
1 - 6 - 4 - 2
```

Below is another example to illustrate the concept of removing duplicates:





Algorithm

The general approach to solve this problem is to loop through the linked list once and keep track of all the data held at each of the nodes. We can use a hash table or Python dictionary to keep track of the data elements that we encounter. For example, if we encounter `6`, we will add that to the dictionary or hash table and move along. Now if we meet another `6` and we check for it in our dictionary or hash table, then we'll know that we already have a `6` and the current node is a duplicate.

Implementation

Let's go ahead and code a solution using the idea discussed above:

```
def remove_duplicates(self):
  cur = self.head
  prev = None
  dup_values = dict()

  while cur:
    if cur.data in dup_values:
      # Remove node:
      prev.next = cur.next
      cur = None
```

```
    else:
      # Have not encountered element before.
      dup_values[cur.data] = 1

      prev = cur
    cur = prev.next
```

remove_duplicates(self)

Explanation

In the `remove_duplicates` method, we'll first declare two variables: `cur` and `prev` and assign them the values `self.head` and `None`, respectively. On **line 4**, we declare a Python dictionary and name it `dup_values`. Now we have to iterate through the linked list using the `while` loop on **line 6**. As you can see, the `while` loop will run until we hit the `None`. Next, we check if `cur.data` exists in `dup_values` or not. Let's first consider the case if `cur.data` does not exist in `dup_values` and move to the `else` portion on **line 11**. We add an entry using `cur.data` as a key to the dictionary and assign `1` as a value to it on **line 13**, while on **line 14**, we update the `prev` with the `cur`.

Now let's move to the case where `cur.data` actually exists from before in the `dup_values`. This is the case where we have found a duplicate! Now we need to remove the duplicate entry. Now, instead of pointing to `cur`, we make `prev.next` point to the next of `cur`, i.e., `cur.next` (**line 9**). Additionally, to completely remove the duplicate entry, i.e. `cur`, we set it equal to `None` on **line 10**.

On **line 15**, we set `cur` to `prev.next` to traverse the linked list.

We have made the `remove_duplicates` method part of the implementation of linked lists. Let's play around and check it on more test cases.

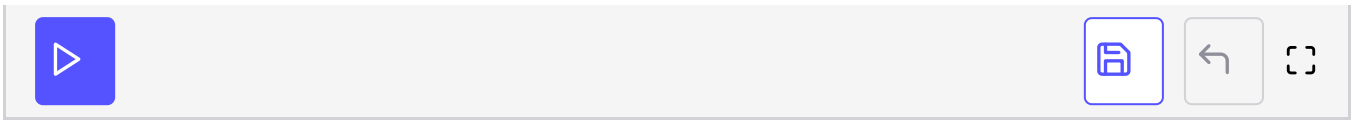
```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None


class LinkedList:
    def __init__(self):
        self.head = None

    def print_list(self):
        cur_node = self.head
        while cur_node:
            print(cur_node.data)
```

```
            cur_node = cur_node.next

    def append(self, data):

        new_node = Node(data)

        if self.head is None:
            self.head = new_node
            return

        last_node = self.head
        while last_node.next:
            last_node = last_node.next
        last_node.next = new_node

    def prepend(self, data):
        new_node = Node(data)

        new_node.next = self.head
        self.head = new_node

    def insert_after_node(self, prev_node, data):

        if not prev_node:
            print("Previous node does not exist.")
            return

        new_node = Node(data)

        new_node.next = prev_node.next
        prev_node.next = new_node

    def delete_node(self, key):

        cur_node = self.head

        if cur_node and cur_node.data == key:
            self.head = cur_node.next
            cur_node = None
            return

        prev = None
        while cur_node and cur_node.data != key:
            prev = cur_node
            cur_node = cur_node.next

        if cur_node is None:
            return

        prev.next = cur_node.next
        cur_node = None

    def delete_node_at_pos(self, pos):
        if self.head:
            cur_node = self.head

            if pos == 0:
                self.head = cur_node.next
                cur_node = None
                return

            prev = None
            count = 1
```

```
        while cur_node and count != pos:
            prev = cur_node
            cur_node = cur_node.next
            count += 1

        if cur_node is None:
            return

        prev.next = cur_node.next
        cur_node = None

    def len_iterative(self):

        count = 0
        cur_node = self.head

        while cur_node:
            count += 1
            cur_node = cur_node.next
        return count

    def len_recursive(self, node):
        if node is None:
            return 0
        return 1 + self.len_recursive(node.next)

    def swap_nodes(self, key_1, key_2):

        if key_1 == key_2:
            return

        prev_1 = None
        curr_1 = self.head
        while curr_1 and curr_1.data != key_1:
            prev_1 = curr_1
            curr_1 = curr_1.next

        prev_2 = None
        curr_2 = self.head
        while curr_2 and curr_2.data != key_2:
            prev_2 = curr_2
            curr_2 = curr_2.next

        if not curr_1 or not curr_2:
            return

        if prev_1:
            prev_1.next = curr_2
        else:
            self.head = curr_2

        if prev_2:
            prev_2.next = curr_1
        else:
            self.head = curr_1

        curr_1.next, curr_2.next = curr_2.next, curr_1.next

    def print_helper(self, node, name):
        if node is None:
            print(name + ": None")
        else:
```

```
        print(name + ":" + node.data)

    def reverse_iterative(self):

        prev = None
        cur = self.head
        while cur:
            nxt = cur.next
            cur.next = prev

            self.print_helper(prev, "PREV")
            self.print_helper(cur, "CUR")
            self.print_helper(nxt, "NXT")
            print("\n")

            prev = cur
            cur = nxt
        self.head = prev

    def reverse_recursive(self):

        def _reverse_recursive(cur, prev):
            if not cur:
                return prev

            nxt = cur.next
            cur.next = prev
            prev = cur
            cur = nxt
            return _reverse_recursive(cur, prev)

        self.head = _reverse_recursive(cur=self.head, prev=None)

    def merge_sorted(self, llist):

        p = self.head
        q = llist.head
        s = None

        if not p:
            return q
        if not q:
            return p

        if p and q:
            if p.data <= q.data:
                s = p
                p = s.next
            else:
                s = q
                q = s.next
            new_head = s
        while p and q:
            if p.data <= q.data:
                s.next = p
                s = p
                p = s.next
            else:
                s.next = q
                s = q
                q = s.next
        if not p:
```

```
            s.next = q
        if not q:
            s.next = p
        return new_head

    def remove_duplicates(self):

        cur = self.head
        prev = None

        dup_values = dict()

        while cur:
            if cur.data in dup_values:
                # Remove node:
                prev.next = cur.next
                cur = None
            else:
                # Have not encountered element before.
                dup_values[cur.data] = 1
                prev = cur
            cur = prev.next

llist = LinkedList()
llist.append(1)
llist.append(6)
llist.append(1)
llist.append(4)
llist.append(2)
llist.append(2)
llist.append(4)

print("Original Linked List")
llist.print_list()
print("Linked List After Removing Duplicates")
llist.remove_duplicates()
llist.print_list()
```



I hope you understood the solution provided in this lesson. See you in the next lesson!