Remove Node

In this lesson, you will learn how to remove a node from a circular linked list using Python.

In this lesson, we investigate how to remove nodes in a circular linked list and code the method in Python.

There is an assumption that we will make before diving into the implementation:

- The occurrences of nodes will be unique, i.e., there will be no duplicate nodes in the circular linked list that we'll test on.

This is because the code that we will write will only be responsible for removing the first occurrence of the key provided to be deleted.

Implementation

Now let's go ahead and jump to the implementation of `remove` in Python:

```
def remove(self, key):
  if self.head:
    if self.head.data == key:
      cur = self.head
      while cur.next != self.head:
        cur = cur.next
      if self.head == self.head.next:
        self.head = None
      else:
        cur.next = self.head.next
        self.head = self.head.next
    else:
      cur = self.head
      prev = None
      while cur.next != self.head:
        prev = cur
        cur = cur.next
        if cur.data == key:
          prev.next = cur.next
          cur = cur.next
```

Explanation

The code is divided into parts based on whether or not we are deleting the head node.

If the condition on **line 2** is `False`, it implies that we are dealing with an empty list and we just return from the method. Otherwise, the execution jumps to **line 3**.

If we are deleting the head node, then the condition on **line 3** will be true, and the execution will jump to **line 4** where we set `cur` equal to `self.head`. To delete the head node, we have to update the node that points to the head node and the node that the head node points to. As a result, we set up a `while` loop on **line 5** which will run until `cur.next` points to `self.head`. We keep updating `cur` to `cur.next` on **line 6**. After the `while` loop terminates, `cur` will be the last node in the linked list which will point to the head node.

At this point, we also need to consider if `self.head` is the only element in the circular linked list. If it is the only element, then `self.head` is pointing to itself. So, we check on **line 7** if that's the case, then `self.head` is set to `None` on **line 8**. On the other hand, if it's not the only element and there is another element to replace the head node, we set `cur.next` equal to `self.head.next` on **line 10**. We have updated the node which was previously pointing to the head node. In the next line, we'll update `self.head` to `self.head.next` which removes the previous head from the linked list and updates the head of the linked list.

Now we'll focus on the else part on **line 12** which refers to the case where we are not deleting the head node. To traverse the linked list and to keep track of the current and previous nodes, we initialize `cur` to `self.head` and `prev` to `None` (**lines 13-14**). On **lines 16-17**, we update `prev` to `cur` and `cur` to `cur.next` to keep track of the nodes in the `while` loop which will traverse the entire linked list once. Next, we have to check for the node to be deleted which we check using the condition on **line 18**. If we find the node to be deleted, we set the next of the previous node (`prev.next`) to the next of the deleted, we set the next of the previous node (`prev.next`) to the next of the

current node (`cur.next`) on **line 19** and then set `cur` to `cur.next` on **line 20** to move along in the linked list.

Below is the entire implementation of the circular linked list that we have covered so far. Feel free to play around with it!

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


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

    def prepend(self, data):
        new_node = Node(data)
        cur = self.head
        new_node.next = self.head

        if not self.head:
            new_node.next = new_node
        else:
            while cur.next != self.head:
                cur = cur.next
            cur.next = new_node
        self.head = new_node

    def append(self, data):
        if not self.head:
            self.head = Node(data)
            self.head.next = self.head
        else:
            new_node = Node(data)
            cur = self.head
            while cur.next != self.head:
                cur = cur.next
            cur.next = new_node
            new_node.next = self.head

    def print_list(self):
        cur = self.head

        while cur:
            print(cur.data)
            cur = cur.next
            if cur == self.head:
                break

    def remove(self, key):
        if self.head:
            if self.head.data == key:
                cur = self.head
                while cur.next != self.head:
                    cur = cur.next
                if self.head == self.head.next:
                    self.head = None

                else:
                    cur.next = self.head.next
                    self.head = self.head.next
            else:
                cur = self.head
                prev = None
                while cur.next != self.head:
                    prev = cur
                    cur = cur.next
                    if cur.data == key:
                        prev.next = cur.next
                        cur = cur.next


cllist = CircularLinkedList()
cllist.append("A")
cllist.append("B")
cllist.append("C")
cllist.append("D")

cllist.remove("A")
cllist.remove("C")
cllist.print_list()
```

CircularLinkedList

In the next lesson, we will look at another problem regarding circular linked lists. See you there!