Solution Review: Deletion by Value

This review provides a detailed analysis of the different ways to solve the Deletion by Value challenge.

We'll cover the following

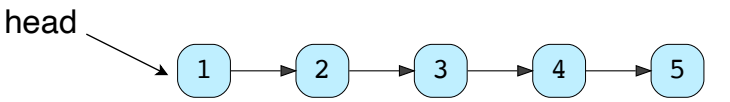
Solution: Search and Delete

main.py

LinkedList.py

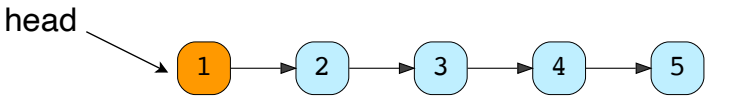
Node.py

```
from LinkedList import LinkedList
from Node import Node


def delete(lst, value):
    deleted = False
    if lst.is_empty():  # Check if list is empty -> Return Fal
        print("List is Empty")
        return deleted
    current_node = lst.get_head()  # Get current node
    previous_node = None  # Get previous node
    if current_node.data is value:
        lst.delete_at_head()  # Use the previous function
        deleted = True
        return deleted

    # Traversing/Searching for Node to Delete
    while current_node is not None:
        # Node to delete is found
        if value is current_node.data:
            # previous node now points to next node
            previous_node.next_element = current_node.next_ele
            current_node.next_element = None
            deleted = True
            break
        previous_node = current_node
        current_node = current_node.next_element

    if deleted is False:
        print(str(value) + " is not in list!")
    else:
        print(str(value) + " deleted!")

    return deleted


lst = LinkedList()
lst.insert_at_head(1)
lst.insert_at_head(4)
lst.insert_at_head(3)
lst.insert_at_head(2)
lst.print_list()
delete(lst, 4)
lst.print_list()

```



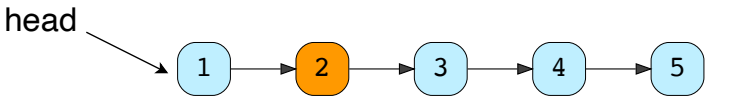
delete(list, 4)

1 of 7

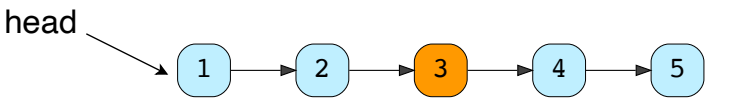


Traverse the list

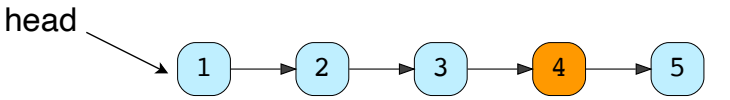
2 of 7



3 of 7

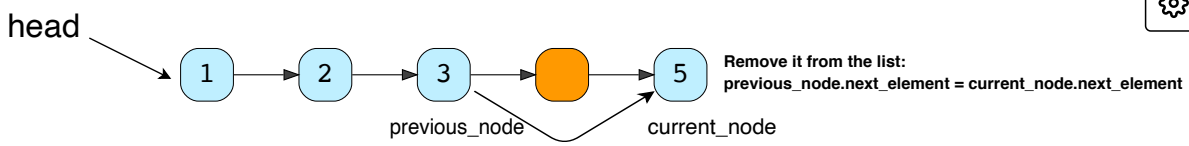


4 of 7

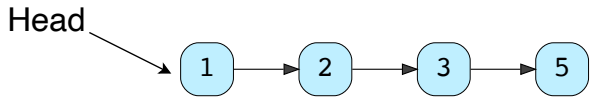


4 found!

5 of 7



6 of 7



Node deleted

7 of 7

The algorithm is very similar to `delete_at_head`. The only difference is that you need to keep track of two nodes, `current_node` and `previous_node`.

`current_node` will always stay one step ahead of `previous_node`. Whenever `current_node` becomes the node to be deleted, the `previous_node` starts pointing at the node next to `current_node`. If `current_node` is the last element, `previous_node` will simply point to `None`.

Congrats! You just implemented the **deletion at tail** strategy as well.

Time Complexity

In the worst case, you would have to traverse until the end of the list. This means the time complexity will be *O(n)*.

So far we have only talked about singly linked lists.

What if our list has bidirectional links? We'll find out more in the next lesson.

Back

Challenge 3: Deletion by Value

Next

Doubly Linked Lists (DLL)

Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/solution-review-deletion-by-value__introduction-to-linked-lists__data-structures-for-coding-interviews-in-python)