educative

Solution Review: Search in a Singly Linked List

This review provides a detailed analysis of the different ways to solve the Search in a Singly Linked List challenge.

We'll cover the following

- Solution: Iterative and Recursive Traversal
    - Time Complexity

Solution: Iterative and Recursive Traversal

Iterative | Recursive

main.py

LinkedList.py

Node.py

```python
from LinkedList import LinkedList
from Node import Node


def search(lst, value):

    # Start from first element
    current_node = lst.get_head()

    # Traverse the list till you reach end
    while current_node:
        if current_node.data is value:
            return True  # value found
        current_node = current_node.next_element

    return False  # value not found


lst = LinkedList()
lst.insert_at_head(4)
lst.insert_at_head(10)
lst.insert_at_head(40)
lst.insert_at_head(5)
lst.print_list()
print(search(lst, 4))

```

In both approaches, we traverse through the list, checking whether the current node’s `data` matches our `value`. The two statements below are equivalent:

```
current_node = current_node.next_element #iterative step

search(node.next_element, value) #recursive step
```

Note that the recursive function takes a node as parameter whereas the iterative version takes the entire list as a parameter.

Time Complexity

The time complexity for this algorithm is *O(n)*. However, the space complexity for the recursive approach is also *O(n)*, whereas the iterative solution can do it in *O(1)* space complexity.

And there you have it. We're done with the **search** operation.

In the next lesson, we will look at how **deletion** works in a singly linked list.

← Back

Challenge 2: Search in a Singly Linked ...

Next →

Singly Linked List Deletion

☑ Completed

Report an Issue

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