Challenge 2: Search in a Singly Linked List

This lesson explains how searching is done in a singly linked list. There is also a coding exercise to test your concepts.

We'll cover the following

Problem Statement

It's time to figure out how to implement another popular linked list function: `search`

To search for a specific value in a linked list, there is no other approach but to traverse the whole list until we find the desired value.

In that sense, the **search** operation in linked lists is similar to the linear search in normal lists or arrays - all of them take *O(n)* amount of time.

The search algorithm in a linked list can be generalized to the following steps:

1. Start from the `head` node.
2. Traverse the list till you either find a node with the given value or you reach the end node which will indicate that the given node doesn't exist in the list.

Input

A linked list and an integer to be searched.

Output

`True` if the integer is found. `False` otherwise.

Sample Input

```
Linked List = 5->90->10->4
Integer = 4
```

Sample Output

```
True
```

Coding Exercise

If you know how to insert an element in a list, then searching for an item will not be that difficult for you.

Now, based on the steps mentioned above, we have designed a simple coding exercise for your practice.

The `Node` and `LinkedList` classes are available to you.

Try to solve this exercise on your own. There are hints to help you along the way.

If you get stuck, you can always refer to the solution which is explained in the next lesson.

main.py

LinkedList.py

Node.py

```
from Node import Node


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

    def get_head(self):
        return self.head_node

    def is_empty(self):
        if(self.head_node is None):  # Check whether the head is None
            return True
        else:
            return False

    # Inserts a value at the end of the list
    def insert_at_tail(self, value):
        # Creating a new node
        new_node = Node(value)

        # Check if the list is empty, if it is simply point head to new node
        if self.get_head() is None:
            self.head_node = new_node
            return

        # if list not empty, traverse the list to the last node
        temp = self.get_head()

        while temp.next_element is not None:
            temp = temp.next_element
```

```
        # Set the nextElement of the previous node to new node
        temp.next_element = new_node
        return

    # Supplementary print function
    def print_list(self):
        if(self.is_empty()):
            print("List is Empty")
            return False
        temp = self.head_node
        while temp.next_element is not None:
            print(temp.data, end=" -> ")
            temp = temp.next_element
        print(temp.data, "-> None")
        return True
```

Searching Node in List

Back

Solution Review: Insertion at Tail

Next

Solution Review: Search in a Singly Li...

Completed

Report an Issue