Challenge 10: Detect Loop in a Linked List

Loops in linked lists can be dangerous as they can end up as programs iterating linked lists indefinitely. Now, you'll create an algorithm to detect them.

We'll cover the following

Problem Statement

By definition, a loop is formed when a node in your linked list points to a previously traversed node.

You must implement the `detect_loop()` function which will take a linked list as input and deduce whether or not a loop is present.

You have already seen this challenge previously in chapter 3 of this course. Here you would use HashTables for a more efficient solution.

Input

A singly linked list.

Output

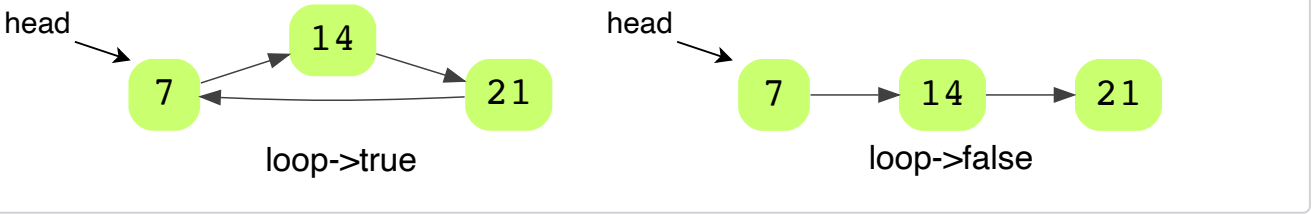
Returns `True` if the given linked list contains a loop. Otherwise, it returns `False`

Sample Input

```
LinkedList = 7->14->21->7 # Both '7's are the same node. Not duplicates.
```

Sample Output

```
True
```



Coding Exercise

There are several ways to implement this function. Flesh out your algorithm and see if it works.

We'll be discussing the most efficient solutions which can be a great help in coding interviews.

As always, the `Node` and `LinkedList` classes are available to you along with all their member functions. If you get stuck, you can always use a hint.

Good luck!

main.py

LinkedList.py

Node.py

```
from LinkedList import LinkedList
from Node import Node
# Access head_node => list.get_head()
# Check if list is empty => list.is_empty()
# Delete at head => list.delete_at_head()
# Delete by value => list.delete(value)
# Search for element => list.search()
# Length of the list => list.length()
# Node class  { int data ; Node next_element;}


def detect_loop(lst):
    # Write your code here
    pass
```


Back

Solution Review: First Non-Repeating ...

Next

Solution Review: Detect Loop in a Link...

Mark as Completed

Report an Issue