Single Linked List practice problems

Adding Two Polynomials Using Linked List

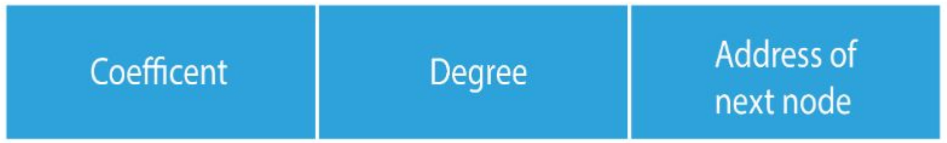
Input:

```
5x4 + 3x2 + 1
4x4 + 2x2 + x
```

Output:

```
9x4 + 5x2 + x + 1
```

Structure of Node



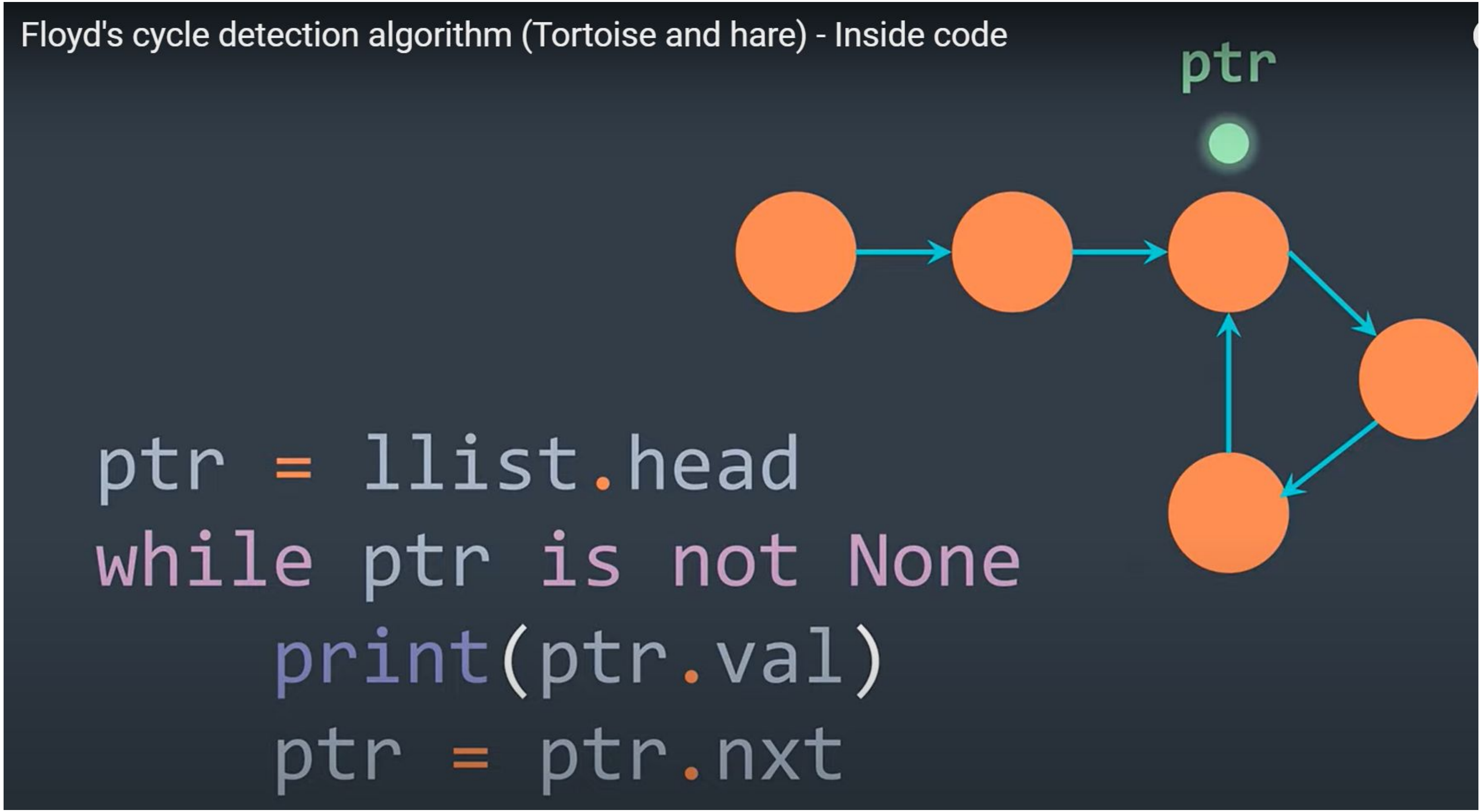
For, this particular problem, The linked list node contains three values:

- **Coefficient:** The numeric value.
- **Degree:** The power of the variable x.
- **Address:** The address of the next node of the linked list.

To add two polynomials that are represented as a linked list, we will add the coefficients of variables with the degree.

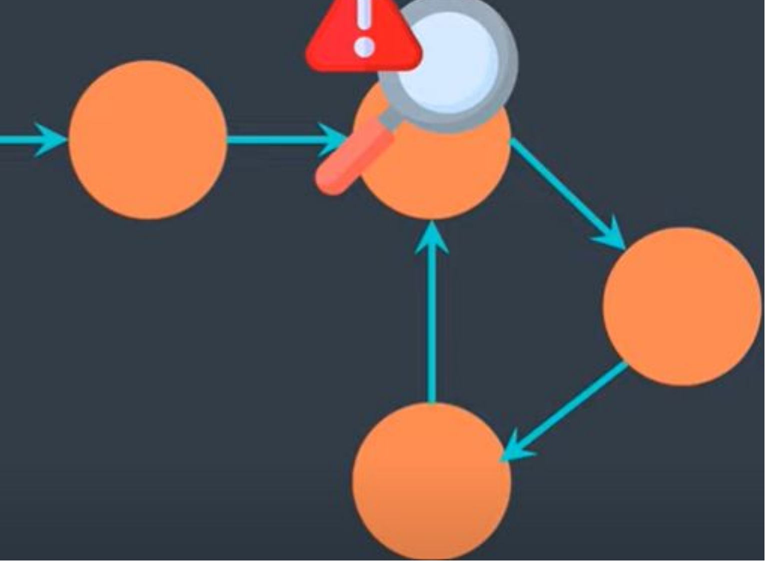
We will traverse both the list and at any step, we will compare the degree of current nodes in both lists:

- We will add their coefficients if their degree is the same and append them to the resultant list.
- Otherwise, we will append the node with a greater node in the resultant list.

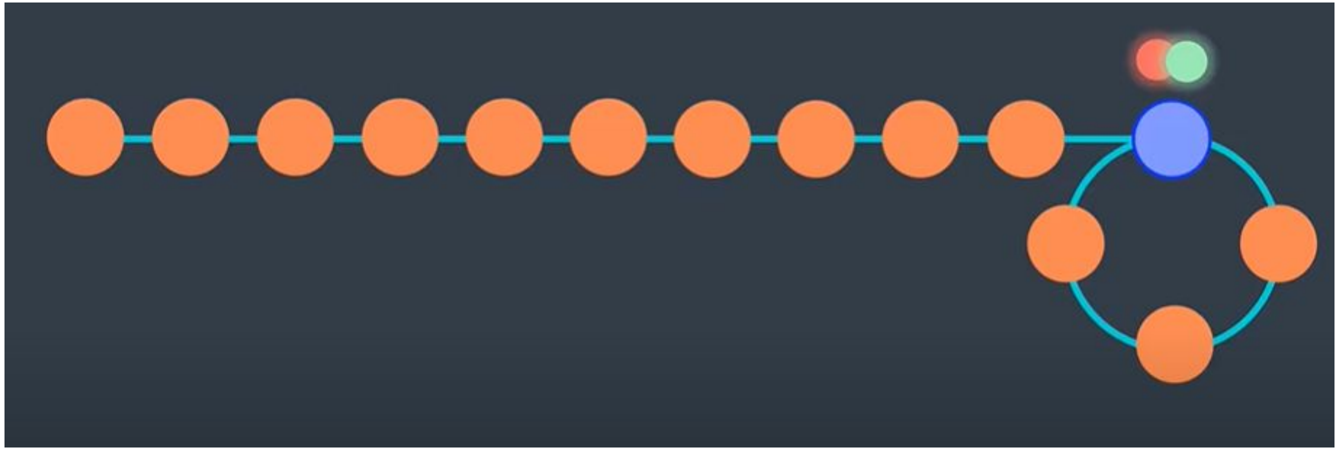


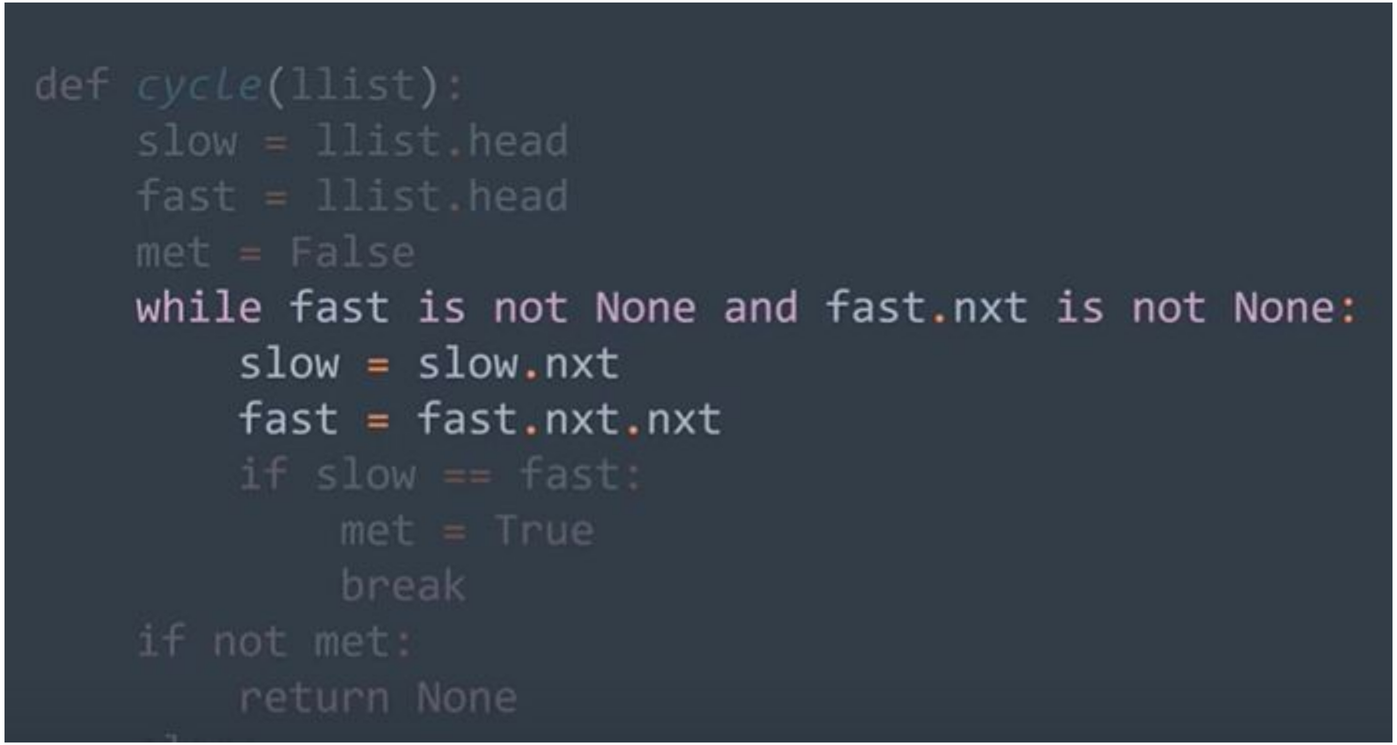
Floyd's cycle detection algorithm (Tortoise and hare) - Inside code

```
ptr = llist.head
while ptr is not None
    print(ptr.val)
    ptr = ptr.nxt
```



Tortoise
Hare





```
def cycle(llist):
    slow = llist.head
    fast = llist.head
    met = False
    while fast is not None and fast.nxt is not None:
        slow = slow.nxt
        fast = fast.nxt.nxt
        if slow == fast:
            met = True
            break
    if not met:
        return None
```

https://youtu.be/PvrxZaH_eZ4?feature=shared

Count duplicates in a given linked list

Goal:

This solution aims to count the number of duplicate nodes in a singly linked list.

Approach:

1. Traverse the linked list.
2. For each node, traverse the remaining part of the list.
3. Check if any nodes in the remaining part of the list have the same data as the current node. If yes, count it as one duplicate and then move on to the next node.

Algorithm:

1. Initialize a variable COUNT to 0. This will keep track of the number of duplicate nodes.
2. For each node in the linked list, referred to as CURRENT_NODE:
   a. Initialize a pointer PTR to the next node of CURRENT_NODE.
   b. While PTR is not NULL:
   i. Compare the data of PTR and CURRENT_NODE.
   ii. If they match, increment COUNT by 1 and break from the inner loop.
   iii. Move PTR to the next node.
3. Continue to the next node of CURRENT_NODE.
4. Once all nodes are processed, return COUNT.

Example:

Let's consider a simple linked list: 3 -> 5 -> 3 -> 7 -> 5.

1. COUNT = 0
2. Start with CURRENT_NODE as the first node (3):
    a. PTR points to 5. No match.
    b. PTR points to 3. Match found. Increment COUNT to 1. Break from inner loop.
3. Move to the next node of CURRENT_NODE, which is 5:
    a. PTR points to 3. No match.
    b. PTR points to 7. No match.
    c. PTR points to 5. Match found. Increment COUNT to 2. Break from inner loop.
4. Move to the next node of CURRENT_NODE, which is the second 3:
    a. PTR points to 7. No match.
    b. PTR points to 5. No match.
    c. End of list for this node. No more matches.
5. Move to the next node of CURRENT_NODE, which is 7:
    a. PTR points to 5. No match.
    b. End of list for this node. No more matches.
6. Move to the next node of CURRENT_NODE, which is the second 5:
    a. End of list for this node. No more matches.
7. COUNT is 2. So, there are 2 nodes in the linked list which have duplicates.