**FELIX OUMA**

**SCT212-0169/2022**

**DATA STRUCTURES AND ALGORITHMS**

**QUIZ 2 – LINKED LISTS**

QUESTION 1

Given head, the head of a linked list, determines if the linked list has a cycle in it. There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer. Internally, pos is used to denote the index of the node that tail's next pointer is connected to. Note that pos is not passed as a parameter.

Return true *if there is a cycle in the linked list*. Otherwise, return false.

\*Python\*

class ListNode:

def \_\_init\_\_(self, x):

self.val = x

self.next = None

def hasCycle(head):

if not head or not head.next:

return False

slow = head

fast = head.next

while slow != fast:

if not fast or not fast.next:

return False

slow = slow.next

fast = fast.next.next

return True

QUESTION 2

Given the head of a linked list, return *the node where the cycle begins. If there is no cycle, return* null. There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer. Internally, pos is used to denote the index of the node that tail's next pointer is connected to (0-indexed). It is -1 if there is no cycle. Note that pos is not passed as a parameter.

**Do not modify the linked list.**

class ListNode:

def \_\_init\_\_(self, x):

self.val = x

self.next = None

def detectCycle(head):

if not head or not head.next:

return None

slow = head

fast = head

# Detect the cycle

while fast and fast.next:

slow = slow.next

fast = fast.next.next

if slow == fast:

break

# If there's no cycle

if not fast or not fast.next:

return None

# Reset one pointer to head and move both at the same pace

slow = head

while slow != fast:

slow = slow.next

fast = fast.next

return slow

QUESTION 3

Write a function that takes the head of a linked list and returns the reversed list.

class ListNode:

def \_\_init\_\_(self, x):

self.val = x

self.next = None

def reverseLinkedList(head):

prev = None

current = head

while current:

next\_node = current.next

current.next = prev

prev = current

current = next\_node

return prev