[Click on the logo to find **Problem Statement**]

Intuition

Imagine two runners running on a track, one is fast (moves two steps at a time) and the other is slow (moves one step at a time). If the track is a circular one, the fast runner will eventually meet the slow runner.

In the context of a linked list:

Cycle Existence: If there is a cycle in the linked list, the fast pointer and slow pointer will eventually meet inside the cycle.

Speed Difference: The fast pointer moves at twice the speed of the slow pointer. If there is a cycle, the fast pointer will "lap" the slow pointer at least once during its traversal.

Meeting Point: Once inside the cycle, the fast and slow pointers are guaranteed to meet. This is because the fast pointer is always closing in on the slow pointer by one node at each step (due to the speed difference). Eventually, they will meet.

No Cycle Case: If there is no cycle, the fast pointer will reach the end of the linked list and the algorithm will terminate. There will be no meeting point of fast and slow pointers in this case.

Approach

Initialize Two Pointers: Start with two pointers, slow and fast, both pointing to the head of the linked list.

Traverse the Linked List: Iterate through the linked list with the slow pointer moving one step at a time and the fast pointer moving two steps at a time. If there is a cycle in the linked list, the fast pointer will eventually meet the slow pointer inside the cycle.

Detect Cycle: If at any point the slow pointer and the fast pointer meet (i.e., slow == fast), then there is a cycle in the linked list. Return True.

End of List: If the fast pointer reaches the end of the list (i.e., fast is None or fast.next is None), then there is no cycle in the linked list. Return False.

Complexity

- Time complexity: O(n)
- Space complexity: O(1)

Code

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def hasCycle(self, head: Optional[ListNode]) -> bool:
        if head is None or head.next is None:
            return False

        slow = head
        fast = head
        while fast is not None and fast.next is not None:
            fast = fast.next.next
            slow = slow.next
            if slow == fast:
                return True
        return False
```