LeetCode 141 — Linked List Cycle

1. Problem Title & Link

- **Title:** LeetCode 141 — Linked List Cycle
- **Link:** https://leetcode.com/problems/linked-list-cycle/

2. Problem Statement (Short Summary)

Given the head of a linked list, determine whether the list contains a **cycle**.

A cycle exists if some node's next pointer points **back** to a previous node.

Return:

- true → cycle found
- false → no cycle

3. Examples (Input → Output)

Example 1

Input: head = [3,2,0,-4], pos = 1

Output: true

(pos indicates index where tail connects → cycle)

Example 2

Input: head = [1,2], pos = 0

Output: true

Example 3

Input: head = [1], pos = -1

Output: false

4. Constraints

- 0 <= nodes <= 10^4
- Node values arbitrary
- List may or may not contain a cycle
- Must use **O(1) space**

5. Core Concept (Pattern / Topic)

Floyd's Cycle Detection Algorithm (Tortoise & Hare)

Two pointers move through the list:

- slow moves 1 step
- fast moves 2 steps

If a cycle exists:

- fast will eventually meet slow.

If no cycle:

- fast will reach null.

6. Thought Process (Step-by-Step Explanation)

Initialize:

slow = head

fast = head

Move:

slow = slow.next

fast = fast.next.next

If at any point slow == fast → cycle detected

If fast becomes null or fast.next becomes null → no cycle

Continue until loop breaks.

Reason:

- In circular movement, the faster pointer eventually “laps” the slower one.

7. Visual / Intuition Diagram

Imagine running on a circular track:

Slow moves 1 step

Fast moves 2 steps

If the track is circular → fast will catch slow.

If the track ends → fast hits null → no cycle.

8. Pseudocode

```
slow = head
fast = head

while fast != null and fast.next != null:
    slow = slow.next
    fast = fast.next.next

    if slow == fast:
        return true

return false
```

9. Code Implementation

✅ **Python**

```
class Solution:
    def hasCycle(self, head: Optional[ListNode]) -> bool:
        slow = fast = head

        while fast and fast.next:
            slow = slow.next
            fast = fast.next.next

            if slow == fast:
                return True

        return False
```

✅ **Java**

```
class Solution {
    public boolean hasCycle(ListNode head) {
        ListNode slow = head, fast = head;

        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;

            if (slow == fast)
                return true;
        }

        return false;
    }
}
```

10. Time & Space Complexity

Metric	Complexity
Time	O(n)
Space	O(1) — constant space

11. Common Mistakes / Edge Cases

Common mistakes

❌ Forgetting to check fast.next before accessing fast.next.next

❌ Thinking cycle means duplicate values (NO — depends on pointers, not values)

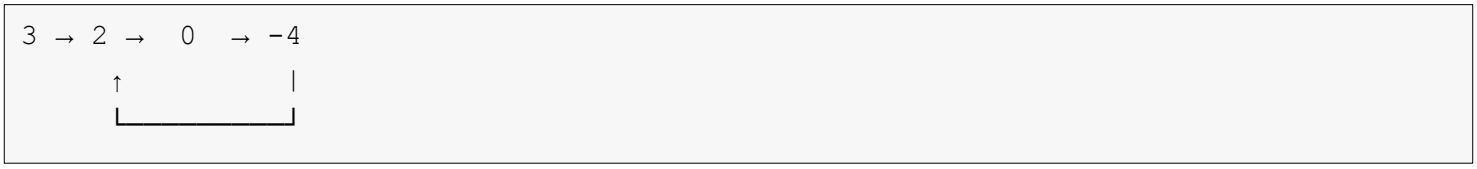
❌ Using a hashset (works but uses O(n) space → not optimal)

Edge cases

- Empty list → no cycle
- Single node without cycle → no cycle
- Single node with cycle → return true

12. Detailed Dry Run (Step-by-Step)

List:



Initial:

slow = head (3)

fast = head (3)

Step 1:

slow = 2

fast = 0

Step 2:

slow = 0

fast = 2

Step 3:

slow = -4

fast = -4 ← meet, cycle exists

Return **true**.

13. Common Use Cases

- Detect loops in:
  - linked lists
  - graph traversal
  - pointer structures
- Deadlock detection in OS
- Cycle detection in function iteration

14. Common Traps

- Using slow-fast incorrectly (incrementing fast by 1 instead of 2)
- Mistaking “node value repeat” with “cycle”
- Losing the list while trying to modify next pointers
- Forgetting to return false when loop exits

15. Builds To (Related Problems)

- **LC 142** — Linked List Cycle II (find starting node of cycle)
- **LC 202** — Happy Number (same slow-fast logic)
- **LC 876** — Middle of Linked List (slow-fast)
- **LC 160** — Intersection of Two Linked Lists

16. Alternate Approaches + Comparison

Approach	Time	Space	Notes
Floyd’s Cycle Detection	O(n)	O(1)	✔ Best
HashSet	O(n)	O(n)	Easy but extra space
Marking nodes	Not allowed	—	Requires node modification

17. Why This Solution Works (Short Intuition)

If two runners run around a circular track,
the faster runner will eventually meet the slower one.
If the track is not circular, the fast runner eventually runs off the path (null).
This maps perfectly to linked lists.

18. Variations / Follow-Up Questions

- Find the **length** of the cycle.
- Remove the cycle and return the list.
- Detect cycle in a graph using DFS vs slow-fast.
- Modify code to return cycle starting node (LC 142).