Documentation: Linked List Cycle Detection

Problem Statement

You are given the head of a linked list. Your task is to determine if the linked list contains a cycle. A linked list is said to have a cycle if there is a node that can be reached again by continuously following the next pointers. The pos variable is used internally to denote the index of the node that the tail's next pointer connects to. Note that pos is not passed as a parameter to the function.

Objective

Write a function to detect if there is a cycle in the linked list.

Definitions

- **Linked List:** A data structure where each element (node) contains a value and a reference (pointer) to the next node in the sequence.
- **Cycle:** A cycle in a linked list occurs when a node's next pointer eventually points back to a previous node in the list, forming a loop.

Input

- **head:** A ListNode object representing the start of the linked list.
- **pos:** An integer where:
  - If pos is -1, the linked list does not contain a cycle.
  - If pos is a valid index, it indicates the position (0-indexed) of the node to which the last node of the list points, creating a cycle.

Output

- Returns true if there is a cycle in the linked list.
- Returns false if there is no cycle in the linked list.

Example 1

- **Input:** head = [3,2,0,-4], pos = 1
- **Output:** true
- **Explanation:** In this case, the last node (-4) points to the node at index 1 (2), creating a cycle.

Example 2

- **Input:** head = [1,2], pos = 0
- **Output:** true
- **Explanation:** Here, the last node (2) points back to the first node (1), forming a cycle.

Example 3

- **Input:** head = [1], pos = -1
- **Output:** false
- **Explanation:** The linked list contains only one node and has no cycle.

Constraints

- The number of nodes in the linked list is within the range [0, 10^4].
- Node values are within the range [-10^5, 10^5].
- pos is either -1 or a valid index within the linked list.

Approach

To solve the problem, you can use the Floyd's Cycle Detection Algorithm (also known as the Tortoise and Hare Algorithm), which is efficient in terms of both time and space complexity.

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

2. **Traverse the List:**

- Move slow by one step and fast by two steps in each iteration.
- If slow and fast meet at the same node, a cycle is detected.

3. **End of List:** If fast reaches the end of the list (i.e., fast or fast.next is None), the list does not contain a cycle.

Follow-up

Can you solve the problem using O(1) (constant) memory? This means avoiding the use of extra space beyond a few pointers.

By using the approach described above, you can detect cycles in a linked list efficiently with constant space complexity.