Documentation for: Convert Sorted List to Binary Search Tree

Problem Statement

Given the head of a singly linked list where elements are sorted in ascending order, convert it to a height-balanced binary search tree (BST).

Example 1:

Input: head = [-10, -3, 0, 5, 9]

Output: [0, -3, 9, -10, null, 5]

Explanation: One possible answer is [0, -3, 9, -10, null, 5], which represents the shown height-balanced BST.

Example 2:

Input: head = []

Output: []

Constraints:

The number of nodes in head is in the range [0, 20,000].

-105 <= Node.val <= 105

Classes

The solution uses two classes to represent the linked list nodes and the binary tree nodes:

- **ListNode:** Represents a node in the linked list.
- **TreeNode:** Represents a node in the binary search tree.

Solution Class

The Solution class contains methods to convert the sorted linked list to a height-balanced BST:

1. findMiddle(self, head):

- Finds the middle node of the linked list.
- Uses a slow pointer (slow_ptr) and a fast pointer (fast_ptr) to traverse the list.
- The fast_ptr moves twice as fast as the slow_ptr. When fast_ptr reaches the end of the list, slow_ptr will be at the middle.
- Keeps track of the previous node (prev_ptr) to slow_ptr. If prev_ptr is not None, it disconnects the left half of the list by setting prev_ptr.next to None.

2. sortedListToBST(self, head):

- Converts the sorted linked list to a height-balanced BST.
- Checks if the list is empty. If it is, it returns None.
- Finds the middle element of the list using the findMiddle method.
- The middle element becomes the root of the BST.
- Recursively forms the left and right subtrees using the left and right halves of the list, respectively.

By following these steps, the solution ensures that the BST is height-balanced. The recursive approach ensures that each subtree is also balanced, resulting in a balanced tree overall.