Problem 173: Binary Search Tree Iterator

Problem Information

Difficulty: Medium
Acceptance Rate: 75.76%
Paid Only: No
Tags: Stack, Tree, Design, Binary Search Tree, Binary Tree, Iterator

Problem Description

Implement the `BSTIterator` class that represents an iterator over the **[in- order traversal](https://en.wikipedia.org/wiki/Tree_traversal#In- order_\(LNR\))** of a binary search tree (BST):

* `BSTIterator(TreeNode root)` Initializes an object of the `BSTIterator` class. The `root` of the BST is given as part of the constructor. The pointer should be initialized to a non-existent number smaller than any element in the BST. * `boolean hasNext()` Returns `true` if there exists a number in the traversal to the right of the pointer, otherwise returns `false`. * `int next()` Moves the pointer to the right, then returns the number at the pointer.

Notice that by initializing the pointer to a non-existent smallest number, the first call to `next()` will return the smallest element in the BST.

You may assume that `next()` calls will always be valid. That is, there will be at least a next number in the in-order traversal when `next()` is called.

Example 1:

![](https://assets.leetcode.com/uploads/2018/12/25/bst-tree.png)

Input ["BSTIterator", "next", "next", "hasNext", "next", "hasNext", "next", "hasNext", "next", "hasNext"] [[[7, 3, 15, null, null, 9, 20]], [], [], [], [], [], [], [], [], []] **Output** [null, 3, 7, true, 9, true, 15, true, 20, false] **Explanation** BSTIterator bSTIterator = new BSTIterator([7, 3, 15, null, null, 9, 20]); bSTIterator.next(); // return 3 bSTIterator.next(); // return 7 bSTIterator.hasNext(); // return True bSTIterator.next(); // return 9 bSTIterator.hasNext(); // return True bSTIterator.next(); // return 15 bSTIterator.hasNext(); // return True bSTIterator.next(); // return 20 bSTIterator.hasNext(); // return False

Constraints:

* The number of nodes in the tree is in the range `[1, 105]`. * `0 <= Node.val <= 106` * At most `105` calls will be made to `hasNext`, and `next`.

Follow up:

* Could you implement `next()` and `hasNext()` to run in average `O(1)` time and use `O(h)` memory, where `h` is the height of the tree?

Code Snippets

C++:

```
/**
* Definition for a binary tree node.
* struct TreeNode {
* int val;
* TreeNode *left;
* TreeNode *right;
* TreeNode() : val(0), left(nullptr), right(nullptr) {}
* TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
* TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
* };
*/
class BSTIterator {
public:
BSTIterator(TreeNode* root) {

}

int next() {

}

bool hasNext() {
}
};

/**
* Your BSTIterator object will be instantiated and called as such:
* BSTIterator* obj = new BSTIterator(root);
* int param_1 = obj->next();
* bool param_2 = obj->hasNext();
*/
```

Java:

```
/**
* Definition for a binary tree node.
* public class TreeNode {
* int val;
* TreeNode left;
* TreeNode right;
* TreeNode() {}
* TreeNode(int val) { this.val = val; }
* TreeNode(int val, TreeNode left, TreeNode right) {
* this.val = val;
* this.left = left;
* this.right = right;
* }
* }
*/
class BSTIterator {

public BSTIterator(TreeNode root) {

}

public int next() {

}

public boolean hasNext() {

}
}
```

```
/**
* Your BSTIterator object will be instantiated and called as such:
* BSTIterator obj = new BSTIterator(root);
* int param_1 = obj.next();
* boolean param_2 = obj.hasNext();
*/
```

Python3:

```
# Definition for a binary tree node.
# class TreeNode:
# def __init__(self, val=0, left=None, right=None):
# self.val = val
# self.left = left
# self.right = right
class BSTIterator:

def __init__(self, root: Optional[TreeNode]):



def next(self) -> int:



def hasNext(self) -> bool:




# Your BSTIterator object will be instantiated and called as such:
# obj = BSTIterator(root)
# param_1 = obj.next()
# param_2 = obj.hasNext()
```