Problem 1586: Binary Search Tree Iterator II

Problem Information

Difficulty: Medium
Acceptance Rate: 63.38%
Paid Only: Yes
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. * `boolean hasPrev()` Returns `true` if there exists a number in the traversal to the left of the pointer, otherwise returns `false`. * `int prev()` Moves the pointer to the left, 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()` and `prev()` calls will always be valid. That is, there will be at least a next/previous number in the in-order traversal when `next()`/`prev()` is called.

Example 1:

Input ["BSTIterator", "next", "next", "prev", "next", "hasNext", "next", "next", "next", "hasNext", "hasPrev", "prev", "prev"] [[[7, 3, 15, null, null, 9, 20]], [null], [null], [null], [null], [null], [null], [null], [null], [null], [null], [null], [null]] **Output** [null, 3, 7, 3, 7, true, 9, 15, 20, false,

true, 15, 9] **Explanation** // The underlined element is where the pointer currently is. BSTIterator bSTIterator = new BSTIterator([7, 3, 15, null, null, 9, 20]); // state is __ [3, 7, 9, 15, 20] bSTIterator.next(); // state becomes [_3_ , 7, 9, 15, 20], return 3 bSTIterator.next(); // state becomes [3, _7_ , 9, 15, 20], return 7 bSTIterator.prev(); // state becomes [_3_ , 7, 9, 15, 20], return 3 bSTIterator.next(); // state becomes [3, _7_ , 9, 15, 20], return 7 bSTIterator.hasNext(); // return true bSTIterator.next(); // state becomes [3, 7, _9_ , 15, 20], return 9 bSTIterator.next(); // state becomes [3, 7, 9, _15_ , 20], return 15 bSTIterator.next(); // state becomes [3, 7, 9, 15, _20_], return 20 bSTIterator.hasNext(); // return false bSTIterator.hasPrev(); // return true bSTIterator.prev(); // state becomes [3, 7, 9, _15_ , 20], return 15 bSTIterator.prev(); // state becomes [3, 7, _9_ , 15, 20], return 9

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`, `next`, `hasPrev`, and `prev`.

Follow up: Could you solve the problem without precalculating the values 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) {

}

bool hasNext() {
```

```
}

int next() {

}

bool hasPrev() {

}

int prev() {

}
};

/**
* Your BSTIterator object will be instantiated and called as such:
* BSTIterator* obj = new BSTIterator(root);
* bool param_1 = obj->hasNext();
* int param_2 = obj->next();
* bool param_3 = obj->hasPrev();
* int param_4 = obj->prev();
*/
```

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 boolean hasNext() {

}

public int next() {

}

public boolean hasPrev() {

}

public int prev() {

}
}

/**
* Your BSTIterator object will be instantiated and called as such:
* BSTIterator obj = new BSTIterator(root);
* boolean param_1 = obj.hasNext();
* int param_2 = obj.next();
* boolean param_3 = obj.hasPrev();
* int param_4 = obj.prev();
*/
```

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 hasNext(self) -> bool:


def next(self) -> int:


def hasPrev(self) -> bool:


def prev(self) -> int:



# Your BSTIterator object will be instantiated and called as such:
# obj = BSTIterator(root)
# param_1 = obj.hasNext()
# param_2 = obj.next()
# param_3 = obj.hasPrev()
# param_4 = obj.prev()
```