173. Binary Search Tree Iterator

03 April 2022 10:32 AM

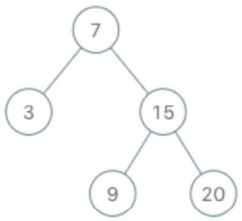
Implement the `BSTIterator` class that represents an iterator over the **in-order traversal** of a binary search tree (BST):

- `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.

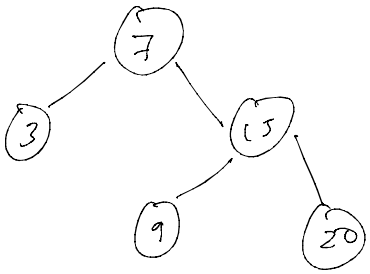
You may assume that `next()` calls will always be valid. That is, there will be at least a next number in the in-order traversal when `next()` is called.

Example 1:




```
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
```



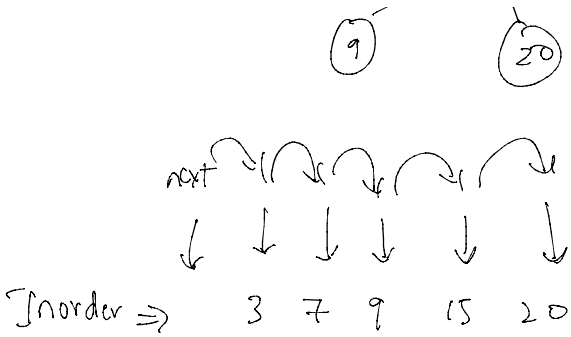
BS iterator (7)

next → 3

next → 7

hm next → True

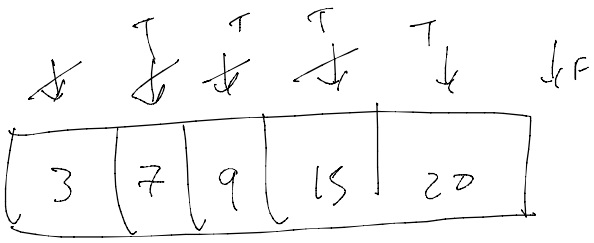
next → 9



hasNext → True
next → 9
hasNext → True
next → 15
hasNext → True
next → 20
hasNext → False

hasnext of 20? = no next
False

1st approach:



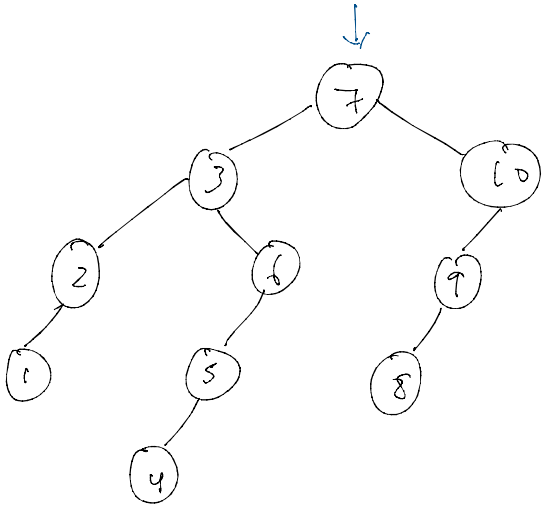
store in array

T.C ⇒ O(1) S.C ⇒ O(N)

↳ Storing entire Inorder Traversal

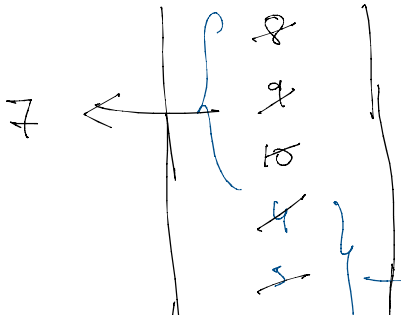
2nd Approach :

Inorder traversal (left root right)



BST iterator (7)

next → 1
next → 2
next → 3
hasnext() → true
next → 4
next → 5
next → 6
next → 7
next → 8
next → 9
next → 10
hasnext → false

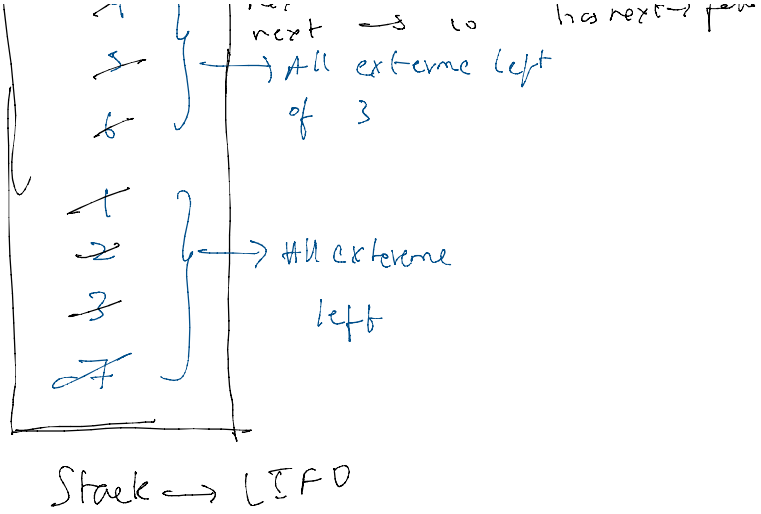


All extreme left

T.C ⇒ O(1)

S.C ⇒ O(h)

pushing all the left elemet




```java
public class BSTIterator {
    private Stack<TreeNode> stack = new Stack<TreeNode>();

    public BSTIterator(TreeNode root) {
        pushAll(root);
    }

    /** @return whether we have a next smallest number */
    public boolean hasNext() {
        return !stack.isEmpty();
    }

    /** @return the next smallest number */
    public int next() {
        TreeNode tmpNode = stack.pop();
        pushAll(tmpNode.right);
        return tmpNode.val;
    }

    private void pushAll(TreeNode node) {
        for (; node != null; stack.push(node), node = node.left);
    }
}
```