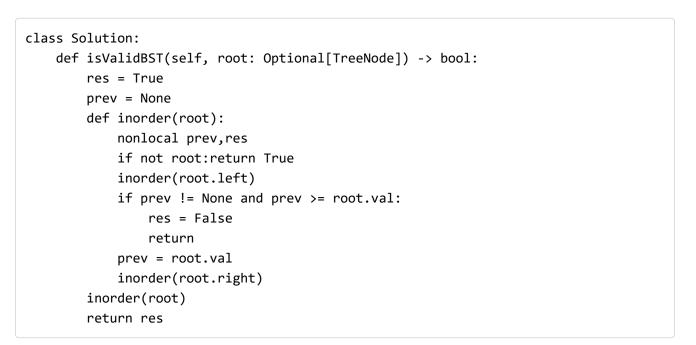
https://leetcode.com/notes/

### 98. Validate Binary Search Tree 🗗



### 99. Recover Binary Search Tree

•

1 of 6 6/14/2025, 7:11 PM

```
class Solution:
    def recoverTree(self, node: Optional[TreeNode]) -> None:
        first,middle,last,prev = None,None,None
        def inorder(root):
            nonlocal first, middle, last, prev
            if not root:
                return None
            inorder(root.left)
            if prev and root.val < prev.val:</pre>
                if not first:
                    first = prev
                    middle = root
                else:
                    last = root
            prev = root
            inorder(root.right)
        inorder(node)
        if first and last:
            first.val,last.val = last.val,first.val
        else:
            middle.val,first.val = first.val,middle.val
        return node
```

### 173. Binary Search Tree Iterator

```
class BSTIterator:
    def fill_stack(self,stack,root):
        while root:
            stack.append(root)
            root = root.left
    def __init__(self, root: Optional[TreeNode]):
        self.root = root
        self.stack = []
        if root:
            self.fill_stack(self.stack,self.root)
    def next(self) -> int:
        node = self.stack.pop()
        if node.right:
            self.fill_stack(self.stack,node.right)
        return node.val
    def hasNext(self) -> bool:
        return True if self.stack else False
```

#### 230. Kth Smallest Element in a BST <sup>C</sup>

```
class Solution:
    def kthSmallest(self, root: Optional[TreeNode], k: int) -> int:
        res = -1
    def inorder(root):
        nonlocal k,res
        if not root or k < 1:
            return
        inorder(root.left)
        if 1 == k:
            res = root.val
        k-=1
        inorder(root.right)
    inorder(root)
    return res</pre>
```

# 235. Lowest Common Ancestor of a Binary Search Tree <sup>17</sup>

```
class Solution:
    def lowestCommonAncestor(self, root: 'TreeNode', p: 'TreeNode', q: 'TreeNode')
-> 'TreeNode':
    if not root:
        return None

cur = root.val
    if p.val < cur and q.val <cur:
        return self.lowestCommonAncestor(root.left,p,q)
    elif p.val > cur and q.val > cur:
        return self.lowestCommonAncestor(root.right,p,q)
    return root
```

### 653. Two Sum IV - Input is a BST

```
class Solution:
    def findTarget(self, root: Optional[TreeNode], k: int) -> bool:
        hashmap = set()
        def inorder(root):
            if not root:
                return False
            if k - root.val in hashmap:
                 return True
            hashmap.add(root.val)
            return inorder(root.left) or inorder(root.right)
        return inorder(root)
```

### 700. Search in a Binary Search Tree 2

4 of 6 6/14/2025, 7:11 PM

```
class Solution:
    def searchBST(self, root: Optional[TreeNode], val: int) -> Optional[TreeNode]:

    def dfs(root,val):
        if not root:return None
        if root.val == val:
            return root
        if val < root.val:
            return dfs(root.left,val)
        else:
            return dfs(root.right,val)
        return None

return dfs(root,val)</pre>
```

### 701. Insert into a Binary Search Tree

## 1008. Construct Binary Search Tree from Preorder Traversal

```
class Solution:
    def bstFromPreorder(self, preorder: List[int]) -> Optional[TreeNode]:
        ind = 0
        def build(bound):
            nonlocal ind
            if ind >= len(preorder):
                return None
            if preorder[ind] <= bound:</pre>
                root = TreeNode(preorder[ind])
            else:
                return None
            ind+=1
            root.left = build(root.val)
            root.right = build(bound)
            return root
        return build(float("inf"))
```