

Problem 1: [Find K-th smallest element in BST](#)

class Node:

```
def __init__(self, value):
```

```
    self.value = value
```

```
    self.left = None
```

```
    self.right = None
```

```
def kth_smallest(root, k):
```

```
    stack = []
```

```
    count = 0
```

```
    curr = root
```

```
    while True:
```

```
        if curr:
```

```
            stack.append(curr)
```

```
            curr = curr.left
```

```
        elif stack:
```

```
            curr = stack.pop()
```

```
            count += 1
```

```
            if count == k:
```

```
                return curr.value
```

```
            curr = curr.right
```

```
        else:
```

```
            break
```

```
root = Node(5)
```

```
root.left = Node(3)
```

```
root.right = Node(7)
```

```
root.left.left = Node(2)
```

```
root.left.right = Node(4)
```

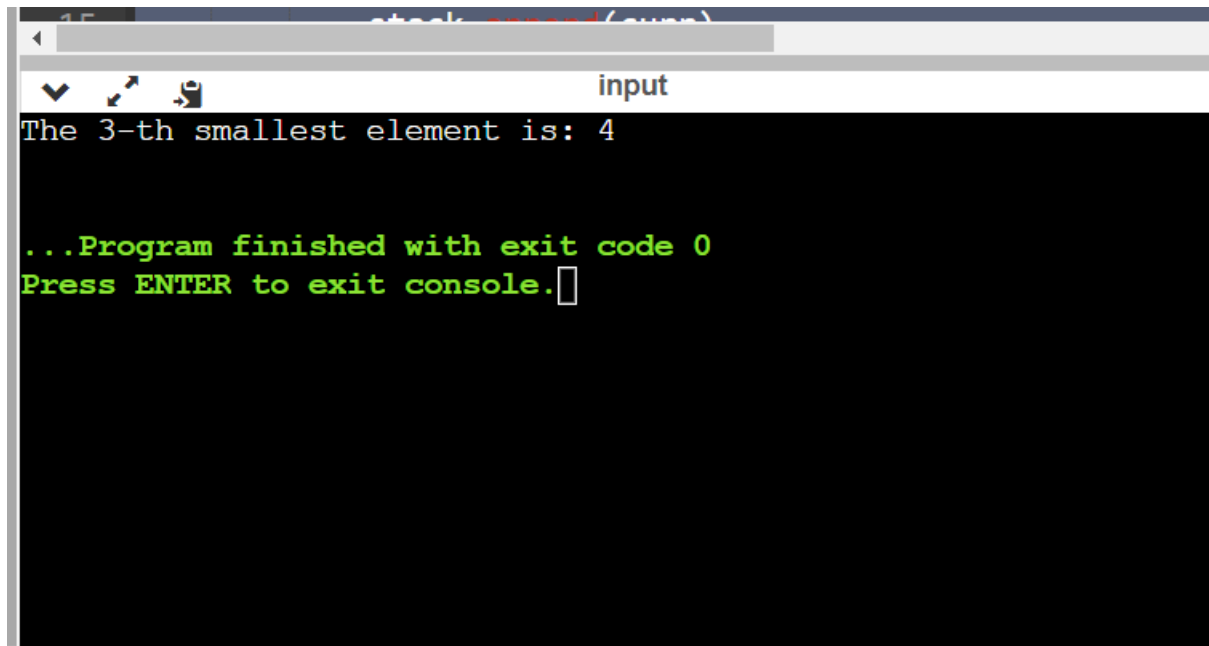
```
root.right.left = Node(6)
```

```
root.right.right = Node(8)
```

```
k = 3
```

```
result = kth_smallest(root, k)
```

```
print(f"The {k}-th smallest element is: {result}")
```

A screenshot of a terminal window. The title bar shows a file icon, a magnifying glass, and the text "input". The terminal content displays the output of a Python program: "The 3-th smallest element is: 4". Below this, in green text, it says "...Program finished with exit code 0" and "Press ENTER to exit console." with a cursor at the end of the line.

```
15 stackoverflow.com  
The 3-th smallest element is: 4  
...Program finished with exit code 0  
Press ENTER to exit console.
```

Problem 2: Ceil in a BST

```
class TreeNode:
```

```
def __init__(self, val=0, left=None, right=None):
```

```
    self.val = val
```

```
    self.left = left
```

```
    self.right = right
```

```
def insert(root, key):
```

```
    if root is None:
```

```
        return TreeNode(key)
```

```
    if key < root.val:
```

```
        root.left = insert(root.left, key)
```

```
    else:
```

```
        root.right = insert(root.right, key)
```

```
    return root
```

```
def ceil(root, target):
```

```
    if root is None:
```

```
        return None
```

```
    if root.val == target:
```

```
        return root.val
```

```
    if root.val < target:
```

```
        return ceil(root.right, target)
```

```
    ceil_val = ceil(root.left, target)
```

```
    if ceil_val is None or ceil_val < target:
```

```
        return root.val
```

```
    return ceil_val
```

```
root = None
```

```
root = insert(root, 8)
```

```
root = insert(root, 4)
```

```
root = insert(root, 12)
```

```
root = insert(root, 2)
```

```
root = insert(root, 6)
```

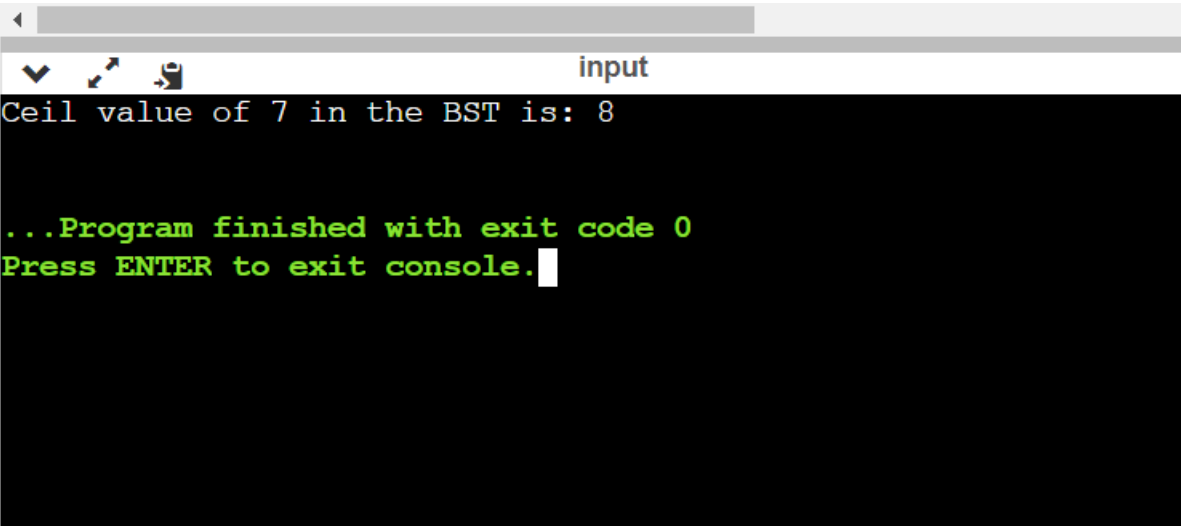
```
root = insert(root, 10)
```

```
root = insert(root, 14)
```

```
target = 7
```

```
ceil_value = ceil(root, target)
```

```
print("Ceil value of", target, "in the BST is:", ceil_value)
```



The screenshot shows a terminal window with a title bar that includes a back arrow, a maximize button, and a close button, followed by the text "input". The terminal content displays the output of the program: "Ceil value of 7 in the BST is: 8" in a light blue monospace font. Below this, in a green monospace font, it says "...Program finished with exit code 0" and "Press ENTER to exit console." with a white cursor at the end of the second line.

Problem 3: Find K-th largest element in BST

```
class TreeNode:
```

```
    def __init__(self, val=0, left=None, right=None):
```

```
        self.val = val
```

```
        self.left = left
```

```
        self.right = right
```

```
def kthLargest(root, k):
```

```
    stack = []
```

```
    node = root
```

```
    while node or stack:
```

```
        while node:
```

```
            stack.append(node)
```

```
            node = node.right
```

```
        node = stack.pop()
```

```
        k -= 1
```

```
        if k == 0:
```

```
            return node.val
```

```
        node = node.left
```

```
root = TreeNode(4)
```

```
root.left = TreeNode(2)
```

```
root.right = TreeNode(7)
```

```
root.left.left = TreeNode(1)
```

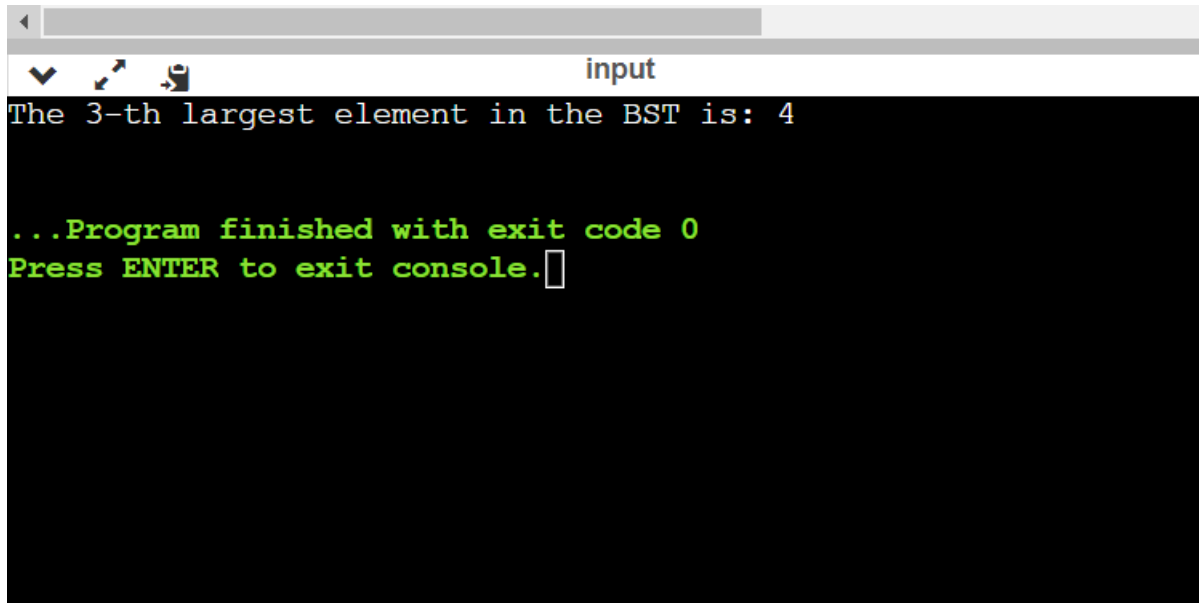
```
root.left.right = TreeNode(3)
```

```
root.right.left = TreeNode(6)
```

k = 3

result = kthLargest(root, k)

print(f"The {k}-th largest element in the BST is: {result}")



The screenshot shows a terminal window with a title bar that includes a back arrow, a maximize button, and a close button, followed by the title "input". The terminal content displays the output of a program: "The 3-th largest element in the BST is: 4". Below this, it shows "...Program finished with exit code 0" and "Press ENTER to exit console." with a cursor at the end of the line.

```
The 3-th largest element in the BST is: 4

...Program finished with exit code 0
Press ENTER to exit console.
```

Problem 4: Find a pair with a given sum in BST

class Node:

```
def __init__(self, val):
```

```
    self.val = val
```

```
    self.left = None
```

```
    self.right = None
```

```
def insert(root, val):
```

```
    if root is None:
```

```
        return Node(val)
```

```
    if val < root.val:
```

```
        root.left = insert(root.left, val)
```

```
    else:
```

```
        root.right = insert(root.right, val)
```

```
    return root
```

```
def in_order_traversal(root, target):
```

```
    stack_left = []
```

```
    stack_right = []
```

```
    curr_left = root
```

```
    curr_right = root
```

```
    done_left = False
```

```
    done_right = False
```

```
    val_left = None
```

```
    val_right = None
```

```
    while True:
```

```
        while not done_left:
```

```
            if curr_left is not None:
```

```
                stack_left.append(curr_left)
```

```
    curr_left = curr_left.left
else:
    if len(stack_left) > 0:
        curr_left = stack_left.pop()
        val_left = curr_left.val
        curr_left = curr_left.right
    else:
        done_left = True
```

```
while not done_right:
    if curr_right is not None:
        stack_right.append(curr_right)
        curr_right = curr_right.right
    else:
        if len(stack_right) > 0:
            curr_right = stack_right.pop()
            val_right = curr_right.val
            curr_right = curr_right.left
        else:
            done_right = True
```

```
if val_left != val_right and val_left + val_right == target:
    return val_left, val_right
```

```
if val_left >= val_right:
    return None
```

```
def find_pair(root, target):
    return in_order_traversal(root, target)
```



```
root = None
```

```
elements = [5, 8, 2, 6, 10]
```

```
for element in elements:
```

```
    root = insert(root, element)
```

```
target_sum = 9
```

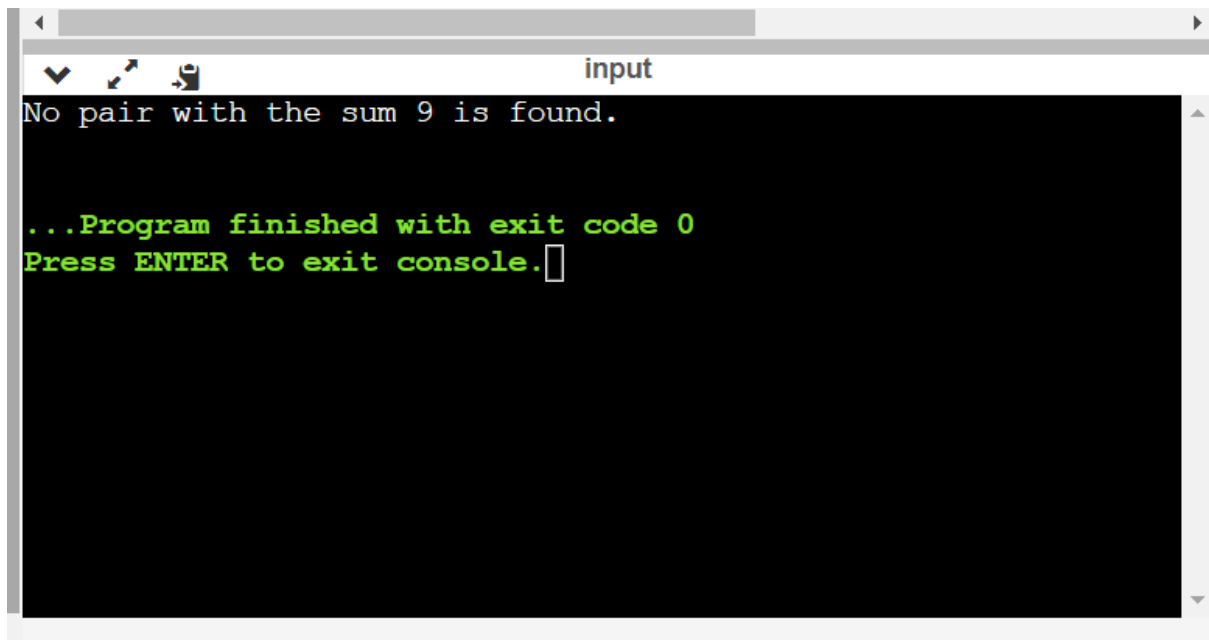
```
pair = find_pair(root, target_sum)
```

```
if pair is not None:
```

```
    print(f"A pair with the sum {target_sum} is found: {pair[0]} and {pair[1]}")
```

```
else:
```

```
    print(f"No pair with the sum {target_sum} is found.")
```



```
input
No pair with the sum 9 is found.

...Program finished with exit code 0
Press ENTER to exit console. □
```

Problem 5: Size of the largest BST in a Binary Tree

```
class Node:
```

```
    def __init__(self, value):
```

```
        self.data = value
```

```
        self.left = None
```

```
        self.right = None
```

```
def largestBSTSize(root):
```

```
    def isBST(node, min_value, max_value):
```

```
        if node is None:
```

```
            return True
```

```
        if node.data < min_value or node.data > max_value:
```

```
            return False
```

```
        return (
```

```
            isBST(node.left, min_value, node.data - 1)
```

```
            and isBST(node.right, node.data + 1, max_value)
```

```
        )
```

```
def countNodes(node):
```

```
    if node is None:
```

```
        return 0
```

```
    return 1 + countNodes(node.left) + countNodes(node.right)
```

```
def largestBSTSizeUtil(node):
```

```
    if isBST(node, float("-inf"), float("inf")):
```

```
        return countNodes(node)
```

```
    return max(
```

```
        largestBSTSizeUtil(node.left),  
        largestBSTSizeUtil(node.right)  
    )
```

```
return largestBSTSizeUtil(root)
```

```
root = Node(6)
```

```
root.left = Node(4)
```

```
root.right = Node(7)
```

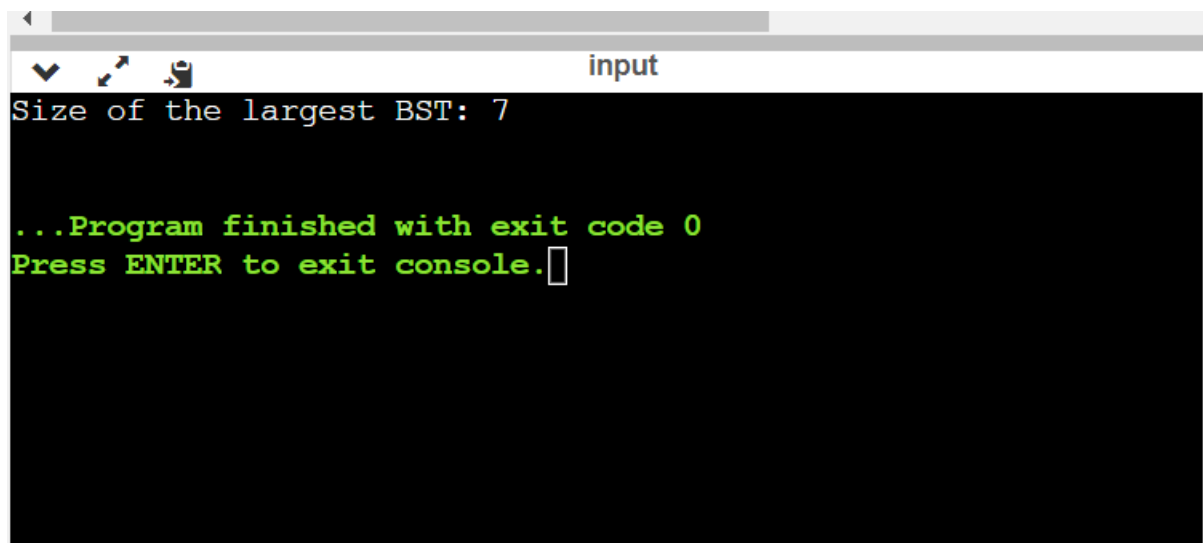
```
root.left.left = Node(3)
```

```
root.left.right = Node(5)
```

```
root.right.right = Node(9)
```

```
root.right.right.left = Node(8)
```

```
print("Size of the largest BST:", largestBSTSize(root))
```



```
Size of the largest BST: 7  
  
...Program finished with exit code 0  
Press ENTER to exit console.□
```

Problem 6: Serialize and deserialize Binary Tree

```
class TreeNode:
```

```
    def __init__(self, value):
```

```
        self.val = value
```

```
        self.left = None
```

```
        self.right = None
```

```
def serialize(root):
```

```
    if not root:
```

```
        return 'None'
```

```
    left_serialized = serialize(root.left)
```

```
    right_serialized = serialize(root.right)
```

```
    return str(root.val) + ',' + left_serialized + ',' + right_serialized
```

```
def deserialize(data):
```

```
    def helper(nodes):
```

```
        if nodes[0] == 'None':
```

```
            nodes.pop(0)
```

```
            return None
```

```
    root = TreeNode(int(nodes[0]))
```

```
nodes.pop(0)

root.left = helper(nodes)

root.right = helper(nodes)
```

```
return root
```

```
nodes = data.split(',')

return helper(nodes)
```

```
root = TreeNode(1)

root.left = TreeNode(2)

root.right = TreeNode(3)

root.right.left = TreeNode(4)

root.right.right = TreeNode(5)
```

```
serialized_tree = serialize(root)

print('Serialized tree:', serialized_tree)
```

```
deserialized_tree = deserialize(serialized_tree)

print('Deserialized tree:', deserialized_tree)
```

```
input
Serialized tree: 1,2,None,None,3,4,None,None,5,None,None
Deserialized tree: <__main__.TreeNode object at 0x7fa4a9a43d00>

...Program finished with exit code 0
Press ENTER to exit console.
```