🎉 Master algorithms together on Binary Search! Create a room, invite your friends, and race to finish the problems.

Daily Coding Problem #307

Problem

This problem was asked by Oracle.

Given a binary search tree, find the floor and ceiling of a given integer. The floor is the highest element in the tree less than or equal to an integer, while the ceiling is the lowest element in the tree greater than or equal to an integer.

If either value does not exist, return None.

Solution

Sometimes it is helpful to compare a problem to a simpler one to gain some intuition.

Suppose that we are trying to determine whether an element exists in a binary search tree. In this case, we can proceed recursively, starting from the root and comparing each node we hit to the value we are searching for. If the value is less than the node data, we search the left child; if the value is greater, we search the right child. Finally, if we reach a leaf node and still have not found the element, we return `None`.

This problem is not too different, in fact. Our recursive function will have two extra parameters, `floor` and `ceil`, which will initially be defined as `None`. At each node, we can update these parameters as follows:

- If `value < node.data`, we know the ceiling can be no greater than `node.data`
- If `value > node.data`, we know the floor can be no less than `node.data`

Once updated, we continue as before, calling our function recursively on the appropriate child node. At the end, when we reach a leaf node, we return the latest and most accurate values for these parameters.

```
class Node:
    def __init__(self, data, left=None, right=None):
        self.data = data
        self.left = left
        self.right = right

def get_bounds(root, x, floor=None, ceil=None):
    if not root:

        return floor, ceil

    if x == root.data:
        return x, x

    elif x < root.data:
        floor, ceil = get_bounds(root.left, x, floor, root.data)

    elif x > root.data:
        floor, ceil = get_bounds(root.right, x, root.data, ceil)

    return floor, ceil
```

This algorithm requires a single traversal from the top to the bottom of the tree. Therefore, the time complexity will be `O(h)`, where `h` is the height of the tree. If the tree is balanced, this is equal to `O(log N)`. Similarly, the space complexity will be `O(h)`, since we will need to make space on the stack for each recursive call.

© Daily Coding Problem 2019

Privacy Policy

Terms of Service

Press