Binary Search Tree traversal - Find Closest Value

Asked 4 months ago Modified 4 months ago Viewed 204 times

0

Im working on a AlgoExpert challenge, I already dedicated time to solving it on my own, watched the video lecture on it and I feel like I have a good understanding, but my skills with recursion and Tree traversal are pretty low right now (that's why I'm working on it).

This is the prompt

> Write a function that takes in a Binary Search Tree (BST) and a target integer value and returns the closest value to that target value contained in the BST. Each BST node has an integer ***value***, a ***left*** child node, and a ***right*** child node. Its children's are valid BST nodes themselves or ***None*** / ***Null***

TARGET: 12

This is my solution so far:

```
function findClosestValueInBst(tree, target) {
    let closest = tree.value;
  const traverse = (inputTree) => {
        if (inputTree === null) return;
        if (Math.abs(target - closest) > Math.abs(target - inputTree.value)) {
            closest = inputTree.value;
        }

        if (target < tree.value) {
            console.log('left')
            traverse(inputTree.left)
        } else {
            console.log('right')
            traverse(inputTree.right)
        }

    }
    traverse(tree)
    return closest;
```

```
}

// This is the class of the input tree. Do not edit.
class BST {
  constructor(value) {
    this.value = value;
    this.left = null;
    this.right = null;
  }
}
```

The behavior so far is, traverse to Node 15, but then, instead of going to 13, it goes to 22 so it returns 10 as the closes possible value instead of 13 which has an absolute value closer to 12 than 10.

javascript algorithm recursion binary-search-tree

Share Improve this question Follow

edited Nov 17, 2021 at 22:12

user1984
5,322 1 8 26

asked Nov 17, 2021 at 21:26

Sebastian Carazo
19 6

2 you are checking against `tree` while the input parameter to the recursive function is `inputTree` – user1984 Nov 17, 2021 at 21:29

you never return the result of `traverse` inside of the function. – Nina Scholz Nov 17, 2021 at 21:31

@NinaScholz, their inner function doesn't return anything. They modify a non-local variable which is returned. – trincot Nov 17, 2021 at 21:33

they don't need to return the result of `traverse` @NinaScholz, as far as I can tell. They are setting a variable that's outside the function. – user1984 Nov 17, 2021 at 21:33

ok, strange design. – Nina Scholz Nov 17, 2021 at 21:36

3 Answers

Sorted by: Highest score (default)

```
function findClosestValueInBst(tree, target) {
    let closest = tree.value;
```

1

```
    const traverse = (inputTree) => {
          if (inputTree === null) return;
          if (Math.abs(target - closest) > Math.abs(target - inputTree.value)) {
              closest = inputTree.value;
          }
          // As you can see below this line you are checking target < tree.value
          // problem is that tree is the root that your surrounding function gets
          // not the argument that your recursive function gets.
          // both your condition and your parameter to traverse
          // need to be inputTree, not tree
          if (target < tree.value) {
              console.log('left')
              traverse(inputTree.left)
          } else {
              console.log('right')
              traverse(inputTree.right)
          }

      }
      traverse(tree)
      return closest;
}
```

Now look at the code below:

```
function findClosestValueInBst(root, target) {
    let closest = root.value;
  const traverse = (node) => {
        if (node === null) return;
        if (Math.abs(target - closest) > Math.abs(target - node.value)) {
            closest = node.value;
        }

        if (target < node.value) {
            console.log('left')
            traverse(node.left)
        } else {
            console.log('right')
            traverse(node.right)
        }

    }
    traverse(root)
```

```
        return closest;
    }
```

In such cases it is helpful to name the parameters more distinct so that no confusion arises.

Share Improve this answer Follow

answered Nov 17, 2021 at 22:06

user1984
5,322 1 8 26

Omg now I see it! I was referencing the wrong tree. thank you! – Sebastian Carazo Nov 17, 2021 at 22:12

No problem. You are welcome :D – user1984 Nov 17, 2021 at 22:12

0

Maybe this works.

It checks the node in a single function and uses the last value with an initial value of a large value.

Then it finds if the target is found and returns the target.

Otherwise check if the target is inside

```
function findClosestValue(tree, target, last = tree.value) {
    if (tree === null) return last;

    if (tree.value === target) return target;

    if (
        last < target && target < tree.value ||
        tree.value < target && target < last
    ) return Math.abs(target - this.value) < Math.abs(target - last)
        ? this.value
        : last;

    return target < tree.value
        ? findClosestValue(tree.left, target, tree.value)
        : findClosestValue(tree.right, target, tree.value);
```

```
}

const
    tree = { value: 10, left: { value: 7, left: { value: 5, left: null, right: null },
right: { value: 8, left: null, right: null } }, right: { value: 13, left: { value: 11,
left: null, right: null }, right: { value: 15, left: null, right: null } } };

console.log(findClosestValue(tree, 11));
console.log(findClosestValue(tree, 12));
console.log(findClosestValue(tree, 13));
console.log(findClosestValue(tree, 14));
console.log(findClosestValue(tree, 15));
console.log(findClosestValue(tree, 16));
console.log(tree);



.as-console-wrapper { max-height: 100% !important; top: 0; }
```

Run code snippet Expand snippet

Share Improve this answer Follow

answered Nov 17, 2021 at 22:23

Nina Scholz

348k 23 296 354

0

Using Nina Scholz's test cases, but what I see as a simpler recursion, we can do this:

```
const closer = (v) => (x, y) =>
  Math.abs (x - v) < Math .abs (y - v) ? x : y

const findClosestValueInBst = (node, target) =>
  node == null
    ? Infinity
  : target == node .value
    ? node .value
  : target < node .value
    ? closer (target) (findClosestValueInBst (node .left, target), node .value)
```

```
  : closer (target) (findClosestValueInBst (node .right, target), node .value)

const tree = { value: 10, left: { value: 7, left: { value: 5, left: null, right: null
}, right: { value: 8, left: null, right: null } }, right: { value: 13, left: { value:
11, left: null, right: null }, right: { value: 15, left: null, right: null } } };

console .log (findClosestValueInBst (tree, 11))
console .log (findClosestValueInBst (tree, 12))
console .log (findClosestValueInBst (tree, 13))
console .log (findClosestValueInBst (tree, 14))
console .log (findClosestValueInBst (tree, 15))
console .log (findClosestValueInBst (tree, 16))

console .log (tree)



.as-console-wrapper {max-height: 100% !important; top: 0}
```

Run code snippet Expand snippet

The `closer` helper simply chooses which of two numbers is closer to a target, arbitrarily choosing the second one if they are equally close.

The main function simply escapes with an infinite value if the tree supplied is `null`, and then branches on whether our `target` is equal to, less than, or greater than the current node's value. If `equal`, we're done and return the node's value. If `less than`, we recur on the left branch and then choose the closer of that result and the node value, and similarly, if `greater than`, we do the same thing with the right branch.

Note that my answers can be different from Nina's because we make different arbitrary choices about whether, say, `11` or `13` is closer to `12`.

Share Improve this answer Follow

answered Nov 17, 2021 at 23:23

Scott Sauyet
43.5k 4 42 92