Closest Binary Search Tree Value

Given a non-empty binary search tree and a target value, find the value in the BST that is closest to the target.

Note:

- Given target value is a floating point.
- You are guaranteed to have only one unique value in the BST that is closest to the target.

Solution 1

Same recursive/iterative solution in different languages.

Recursive

Closest is either the root's value (`a`) or the closest in the appropriate subtree (`b`).

Ruby

```ruby
def closest_value(root, target)
  a = root.val
  kid = target < a ? root.left : root.right or return a
  b = closest_value(kid, target)
  [b, a].min_by { |x| (x - target).abs }
end
```

C++

```cpp
int closestValue(TreeNode* root, double target) {
    int a = root->val;
    auto kid = target < a ? root->left : root->right;
    if (!kid) return a;
    int b = closestValue(kid, target);
    return abs(a - target) < abs(b - target) ? a : b;
}
```

Java

```java
public int closestValue(TreeNode root, double target) {
    int a = root.val;
    TreeNode kid = target < a ? root.left : root.right;
    if (kid == null) return a;
    int b = closestValue(kid, target);
    return Math.abs(a - target) < Math.abs(b - target) ? a : b;
}
```

Python

```python
def closestValue(self, root, target):
    a = root.val
    kid = root.left if target < a else root.right
    if not kid: return a
    b = self.closestValue(kid, target)
    return min((b, a), key=lambda x: abs(target - x))
```

Alternative endings:

```python
    return (b, a)[abs(a - target) < abs(b - target)]
    return a if abs(a - target) < abs(b - target) else b
```

Iterative

Walk the path down the tree close to the target, return the closest value on the path. Inspired by yd, I wrote these after reading "while loop".

Ruby

```ruby
def closest_value(root, target)
  path = []
  while root
    path << root.val
    root = target < root.val ? root.left : root.right
  end
  path.reverse.min_by { |x| (x - target).abs }
end
```

The `.reverse` is only for handling targets much larger than 32-bit integer range, where different path values x have the same "distance" `(x - target).abs`. In such cases, the leaf value is the correct answer. If such large targets aren't asked, then it's unnecessary.

Or with O(1) space:

```ruby
def closest_value(root, target)
  closest = root.val
  while root
    closest = [root.val, closest].min_by { |x| (x - target).abs }
    root = target < root.val ? root.left : root.right
  end
  closest
end
```

C++

```cpp
int closestValue(TreeNode* root, double target) {
    int closest = root->val;
    while (root) {
        if (abs(closest - target) >= abs(root->val - target))
            closest = root->val;
        root = target < root->val ? root->left : root->right;
    }
    return closest;
}
```

Python

```
def closestValue(self, root, target):
    path = []
    while root:
        path += root.val,
        root = root.left if target < root.val else root.right
    return min(path[::-1], key=lambda x: abs(target - x))
```

The `[::-1]` is only for handling targets much larger than 32-bit integer range, where different path values x have the same "distance" `(x - target).abs`. In such cases, the leaf value is the correct answer. If such large targets aren't asked, then it's unnecessary.

Or with O(1) space:

```
def closestValue(self, root, target):
    closest = root.val
    while root:
        closest = min((root.val, closest), key=lambda x: abs(target - x))
        root = root.left if target < root.val else root.right
    return closest
```

written by StefanPochmann original link here

Solution 2

```
public int closestValue(TreeNode root, double target) {
    int ret = root.val;
    while(root != null){
        if(Math.abs(target - root.val) < Math.abs(target - ret)){
            ret = root.val;
        }
        root = root.val > target? root.left: root.right;
    }
    return ret;
}
```

written by larrywang2014 original link here

Solution 3

```
public int closestValue(TreeNode root, double target) {
        int closestVal = root.val;
        while(root != null){
            //update closestVal if the current value is closer to target
            closestVal = (Math.abs(target - root.val) < Math.abs(target - closest
Val))? root.val : closestVal;
            if(closestVal == target){   //already find the best result
                return closestVal;
            }
            root = (root.val > target)? root.left: root.right;   //binary search
        }
        return closestVal;
  }
```

written by ranylee2 original link here