Tian Qiu

00265 35063

CS 381 HW 3



Procedure:

// low and high value start from the different side

If root.value > low && root.value < high:

Result = root.total // root is the total value

1. From root, use binary search to find the first value that is smaller than low value,

denoting as first\_low\_node.

1. From root, use binary search to find the first value that is larger than high value,

denoting as first\_high\_node.

1. Result = Result - first\_low\_node->left.total // O (1)

d. Result = Result - first\_high\_node->right.total // O (1)

1. Recur the procedure (a) (c) for first\_low\_node->right, // O (log n)

until first\_low\_node->left or first\_low\_node->right is null.

f. Recur the procedure (b) (d) for first\_low\_node->left, // O (log n)

until first\_low\_node->left or first\_low\_node->right is null.

// low and high value start from the same side

Else:

1. From root, use binary search to find the first value that is larger than high value,

denoting as first\_high\_node.

1. From root, use binary search to find the first value that is smaller than low value,

denoting as first\_low\_node.

// first result value cannot be from root since low and high value in the same side

// So we have to determine the way to assign result

if first\_low\_node is the parent of first\_high\_node:

result = first\_low\_node.total

else if first\_high\_node is the parent of first\_low\_node:

result = first\_high\_node.total

else

co\_parent = the first co-parent of both first high and low node

result = co\_parent.total

1. Result = Result - first\_low\_node->left.total // O (1)
2. Result = Result - first\_high\_node->right.total // O (1)
3. Recur the procedure (a) (c) for first\_low\_node->right, // O (log n)

until first\_low\_node->left or first\_low\_node->right is null.

1. Recur the procedure (b) (d) for first\_low\_node->left, // O (log n)

until first\_low\_node->left or first\_low\_node->right is null.

Return result

Time analysis:

To search and do the minus operation for the first\_low\_node and first\_high\_node for each subtree, the worst case is to go from top to the bottom which is O (log n). Also if the first\_low\_node and first\_high\_node do not occur in the same side, we have to find co parent node of them, which is also O (log n) in the worst case.

So T(n) = 2 \* ( + ) + = 5 = O (log n)



A).

the first and second line ought to be revised as:

r = x.left.size + x.count

if x.left.size < i && i <=r:

return x

because we have to make sure the x is going down but not stay in the same level.

B).

When we insert we have to add one more if statement as following:

If node.val == insert\_val: // if the value of the node already exists

Node.count ++