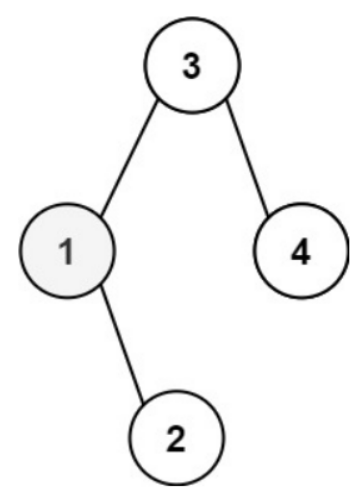
<https://leetcode.com/problems/kth-smallest-element-in-a-bst/>

Given the root of a binary search tree, and an integer k, return *the* kth *smallest value (****1-indexed****) of all the values of the nodes in the tree*.

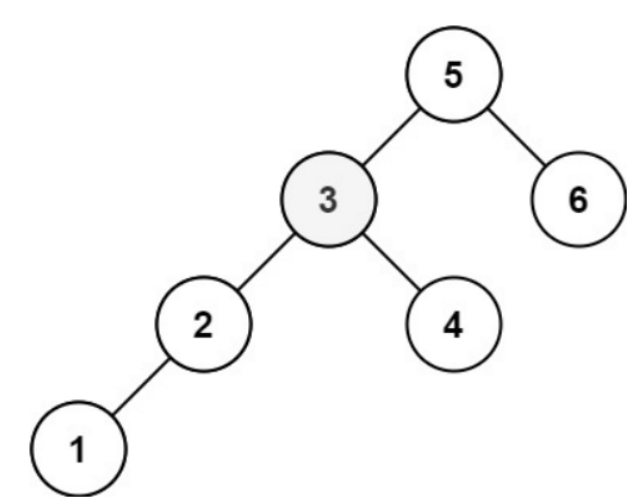
**Example 1:**



Input: root = [3,1,4,null,2], k = 1

Output: 1

**Example 2:**



Input: root = [5,3,6,2,4,null,null,1], k = 3

Output: 3

**Constraints:**

* The number of nodes in the tree is n.
* 1 <= k <= n <= 104
* 0 <= Node.val <= 104

**Follow up:** If the BST is modified often (i.e., we can do insert and delete operations) and you need to find the kth smallest frequently, how would you optimize?

**Attempt 1: 2022-12-17**

**Solution 1: Recursive traversal (10 min)**

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode() {}

\* TreeNode(int val) { this.val = val; }

\* TreeNode(int val, TreeNode left, TreeNode right) {

\* this.val = val;

\* this.left = left;

\* this.right = right;

\* }

\* }

\*/

class Solution {

int count = 0;

int result = Integer.MIN\_VALUE;

public int kthSmallest(TreeNode root, int k) {

count = k;

helper(root);

return result;

}

private void helper(TreeNode root) {

if(root == null) {

return;

}

helper(root.left);

count--;

if(count == 0) {

result = root.val;

// Find result early return

return;

}

helper(root.right);

}

}

Time Complexity : O(N)

Space Complexity : O(N)

**Solution 2: Iterative traversal  (10 min)**

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode() {}

\* TreeNode(int val) { this.val = val; }

\* TreeNode(int val, TreeNode left, TreeNode right) {

\* this.val = val;

\* this.left = left;

\* this.right = right;

\* }

\* }

\*/

class Solution {

public int kthSmallest(TreeNode root, int k) {

int count = k;

Stack<TreeNode> stack = new Stack<TreeNode>();

while(root != null || !stack.isEmpty()) {

while(root != null) {

stack.push(root);

root = root.left;

}

root = stack.pop();

count--;

if(count == 0) {

return root.val;

}

root = root.right;

}

return Integer.MIN\_VALUE;

}

}

Time Complexity : O(N)

Space Complexity : O(N)