

Second Minimum Node In a Binary Tree

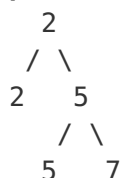
Given a non-empty special binary tree consisting of nodes with the non-negative value, where each node in this tree has exactly **two** or **zero** sub-node. If the node has two sub-nodes, then this node's value is the smaller value among its two sub-nodes.

Given such a binary tree, you need to output the **second minimum** value in the set made of all the nodes' value in the whole tree.

If no such second minimum value exists, output -1 instead.

Example 1:

Input:

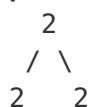


Output: 5

Explanation: The smallest value is 2, the second smallest value is 5.

Example 2:

Input:



Output: -1

Explanation: The smallest value is 2, but there isn't any second smallest value.

Solution 1

This question is very similar to searching for minimum value in the Binary Tree. The only requirement is that this value must be different from the root value, k.

```
If the root value of a subtree == k,  
    keep searching its children.  
else,  
    return the root value because it is the minimum of current subtree.
```

```
class Solution {  
public:  
    int findSecondMinimumValue(TreeNode* root) {  
        if (!root) return -1;  
        int ans = minval(root, root->val);  
        return ans;  
    }  
private:  
    int minval(TreeNode* p, int first) {  
        if (p == nullptr) return -1;  
        if (p->val != first) return p->val;  
        int left = minval(p->left, first), right = minval(p->right, first);  
        // if all nodes of a subtree = root->val,  
        // there is no second minimum value, return -1  
        if (left == -1) return right;  
        if (right == -1) return left;  
        return min(left, right);  
    }  
};
```

written by [zestypanda](#) original link [here](#)

Solution 2

```
public int findSecondMinimumValue(TreeNode root) {
    if (root == null) {
        return -1;
    }
    Set<Integer> set = new TreeSet<>();
    dfs(root, set);
    Iterator<Integer> iterator = set.iterator();
    int count = 0;
    while (iterator.hasNext()) {
        count++;
        int result = iterator.next();
        if (count == 2) {
            return result;
        }
    }
    return -1;
}

private void dfs(TreeNode root, Set<Integer> set) {
    if (root == null) {
        return;
    }
    set.add(root.val);
    dfs(root.left, set);
    dfs(root.right, set);
    return;
}
```

Also viewable [here](#) on Github.

written by [fishercoder](#) original link [here](#)

Solution 3

Java:

```
public int findSecondMinimumValue(TreeNode root)
{
    int rootVal = root.val;
    int secondSmall = Integer.MAX_VALUE;
    boolean diffFound = false;
    Queue<TreeNode> Q= new LinkedList<TreeNode>();
    Q.add(root);

    while(!Q.isEmpty())
    {
        TreeNode curr=Q.poll();
        if(curr.val!=rootVal && curr.val < secondSmall)
        {
            secondSmall=curr.val;
            diffFound=true;
        }
        if(curr.left!=null)
        {
            Q.add(curr.left);
            Q.add(curr.right);
        }
    }

    return (secondSmall == Integer.MAX_VALUE && !diffFound) ? -1 : secondSmall;
}
```

C#:

```
public int FindSecondMinimumValue(TreeNode root)
{
    int rootVal = root.val;
    int secondSmall = int.MaxValue;
    bool diffFound = false;
    Queue<TreeNode> Q = new Queue<TreeNode>();
    Q.Enqueue(root);

    while (Q.Any()) //while Q is not empty
    {
        TreeNode curr = Q.Dequeue();
        if (curr.val != rootVal && curr.val <= secondSmall)
        {
            secondSmall = curr.val;
            diffFound = true;
        }
        if (curr.left != null)
        {
            Q.Enqueue(curr.left);
            Q.Enqueue(curr.right);
        }
    }
    return (secondSmall == int.MaxValue && !diffFound) ? -1 : secondSmall;
}
```

written by [yashar](#) original link [here](#)

From [Leetcode](#).