

Minimum Absolute Difference in BST

Given a binary search tree with non-negative values, find the minimum **absolute difference** between values of any two nodes.

Example:

Input:

```
  1
   \
    3
   /
  2
```

Output:

1

Explanation:

The minimum absolute difference is 1, which is the difference between 2 and 1 (or between 2 and 3).

Note: There are at least two nodes in this BST.

Solution 1

The most common idea is to first **inOrder** traverse the tree and compare the delta between each of the adjacent values. It's guaranteed to have the correct answer because it is a **BST** thus **inOrder** traversal values are **sorted**.

Solution 1 - In-Order traverse, time complexity $O(N)$, space complexity $O(1)$.

```
public class Solution {
    int min = Integer.MAX_VALUE;
    Integer prev = null;

    public int getMinimumDifference(TreeNode root) {
        if (root == null) return min;

        getMinimumDifference(root.left);

        if (prev != null) {
            min = Math.min(min, root.val - prev);
        }
        prev = root.val;

        getMinimumDifference(root.right);

        return min;
    }
}
```

What if it is not a **BST**? (Follow up of the problem) The idea is to put values in a **TreeSet** and then every time we can use **$O(\lg N)$** time to lookup for the nearest values.

Solution 2 - Pre-Order traverse, time complexity $O(N \lg N)$, space complexity $O(N)$.

```
public class Solution {
    TreeSet<Integer> set = new TreeSet<>();
    int min = Integer.MAX_VALUE;

    public int getMinimumDifference(TreeNode root) {
        if (root == null) return min;

        if (!set.isEmpty()) {
            if (set.floor(root.val) != null) {
                min = Math.min(min, Math.abs(root.val - set.floor(root.val)));
            }
            if (set.ceiling(root.val) != null) {
                min = Math.min(min, Math.abs(root.val - set.ceiling(root.val)));
            }
        }

        set.add(root.val);

        getMinimumDifference(root.left);
        getMinimumDifference(root.right);

        return min;
    }
}
```

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

Solution 2

In-order traversal of BST yields sorted sequence. So, we just need to subtract the previous element from the current one, and keep track of the minimum. We need $O(1)$ memory as we only store the previous element, but we still need $O(h)$ for the stack.

```
void inorderTraverse(TreeNode* root, int& val, int& min_dif) {
    if (root->left != NULL) inorderTraverse(root->left, val, min_dif);
    if (val >= 0) min_dif = min(min_dif, root->val - val);
    val = root->val;
    if (root->right != NULL) inorderTraverse(root->right, val, min_dif);
}
int getMinimumDifference(TreeNode* root) {
    auto min_dif = INT_MAX, val = -1;
    inorderTraverse(root, val, min_dif);
    return min_dif;
}
```

Another solution with the member variables (6 lines):

```
class Solution {
    int min_dif = INT_MAX, val = -1;
public:
    int getMinimumDifference(TreeNode* root) {
        if (root->left != NULL) getMinimumDifference(root->left);
        if (val >= 0) min_dif = min(min_dif, root->val - val);
        val = root->val;
        if (root->right != NULL) getMinimumDifference(root->right);
        return min_dif;
    }
}
```

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

Solution 3

Since this is a BST, the inorder traversal of its nodes results in a sorted list of values. Thus, the minimum absolute difference must occur in any adjacently traversed nodes. I use the global variable "prev" to keep track of each node's inorder predecessor.

```
public class Solution {  
  
    int minDiff = Integer.MAX_VALUE;  
    TreeNode prev;  
  
    public int getMinimumDifference(TreeNode root) {  
        inorder(root);  
        return minDiff;  
    }  
  
    public void inorder(TreeNode root) {  
        if (root == null) return;  
        inorder(root.left);  
        if (prev != null) minDiff = Math.min(minDiff, root.val - prev.val);  
        prev = root;  
        inorder(root.right);  
    }  
}
```

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

From [LeetCoder](#).