<https://www.lintcode.com/problem/900/>

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

* 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.

**Example**

**Example1**

Input: root = {5,4,9,2,#,8,10} and target = 6.124780

Output: 5

Explanation：

Binary tree {5,4,9,2,#,8,10}, denote the following structure:

5

/ \

4 9

/ / \

2 8 10

**Example2**

Input: root = {3,2,4,1} and target = 4.142857

Output: 4

Explanation：

Binary tree {3,2,4,1}, denote the following structure:

3

/ \

2 4

/

1

**Attempt 1: 2022-12-09**

**Solution 1:  Classic Inorder Recursive Traversal (30 min, be careful on Math.abs(...) return double)**

**Style 1: With additional list to store inoder recursive traversal result**

/\*\*

\* Definition of TreeNode:

\* public class TreeNode {

\* public int val;

\* public TreeNode left, right;

\* public TreeNode(int val) {

\* this.val = val;

\* this.left = this.right = null;

\* }

\* }

\*/

public class Solution {

/\*\*

\* @param root: the given BST

\* @param target: the given target

\* @return: the value in the BST that is closest to the target

\*/

public int closestValue(TreeNode root, double target) {

List<Integer> list = new ArrayList<Integer>();

inorder(root, list);

int result = 0;

double minDelta = Double.MAX\_VALUE;

for(int num : list) {

if(Math.abs(num - target) < minDelta) {

result = num;

minDelta = Math.abs(num - target);

}

}

return result;

}

private void inorder(TreeNode root, List<Integer> list) {

if(root == null) {

return;

}

inorder(root.left, list);

list.add(root.val);

inorder(root.right, list);

}

}

Time Complexity: O(N)

Space Complexity: O(N)

**Refer to**

<https://wentao-shao.gitbook.io/leetcode/binary-tree/270.closest-binary-search-tree-value>

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

class Solution {

public int closestValue(TreeNode root, double target) {

List<Integer> nums = new ArrayList();

inorder(root, nums);

return Collections.min(nums, new Comparator<Integer>() {

@Override

public int compare(Integer o1, Integer o2) {

return Math.abs(o1 - target) - Math.abs(o2 - target);

}

});

}

private void inorder(TreeNode root, List<Integer> nums) {

if (root == null) return;

inorder(root.left, nums);

nums.add(root.val);

inorder(root.right, nums);

}

}

**Style 2: Without additional list to store inoder recursive traversal result, adding global variable**

**Note: the underlying logic is simple traverse each node on tree (inorder, preorder and postorder all works), when visiting each node, at the same time calculate its delta against 'target' , to find the node has minimum delta against 'target', we need to initialize a global variable 'closet' to record at which node we get the minimum delta.**

/\*\*

\* Definition of TreeNode:

\* public class TreeNode {

\* public int val;

\* public TreeNode left, right;

\* public TreeNode(int val) {

\* this.val = val;

\* this.left = this.right = null;

\* }

\* }

\*/

public class Solution {

/\*\*

\* @param root: the given BST

\* @param target: the given target

\* @return: the value in the BST that is closest to the target

\*/

int closet = Integer.MIN\_VALUE;

public int closestValue(TreeNode root, double target) {

inorder(root, target);

return closet;

}

private void inorder(TreeNode root, double target) {

if(root == null) {

return;

}

inorder(root.left, target);

if(Math.abs(root.val - target) < Math.abs(target - closet)) {

closet = root.val;

}

inorder(root.right, target);

}

}

Time Complexity: O(N)

Space Complexity: O(1)

**Solution 2:  Classic Inorder Iterative Traversal (10 min)**

**Style 1: With additional list to store inorder iterative traversal result**

/\*\*

\* Definition of TreeNode:

\* public class TreeNode {

\* public int val;

\* public TreeNode left, right;

\* public TreeNode(int val) {

\* this.val = val;

\* this.left = this.right = null;

\* }

\* }

\*/

public class Solution {

/\*\*

\* @param root: the given BST

\* @param target: the given target

\* @return: the value in the BST that is closest to the target

\*/

public int closestValue(TreeNode root, double target) {

List<Integer> list = new ArrayList<Integer>();

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

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

while(root != null) {

stack.push(root);

root = root.left;

}

root = stack.pop();

list.add(root.val);

root = root.right;

}

int result = 0;

double minDelta = Double.MAX\_VALUE;

for(int num : list) {

if(Math.abs(num - target) < minDelta) {

result = num;

minDelta = Math.abs(num - target);

}

}

return result;

}

}

Time Complexity : O(N)

Space Complexity: O(N)

**Style 2: Without additional list to store inorder iterative traversal result**

/\*\*

\* Definition of TreeNode:

\* public class TreeNode {

\* public int val;

\* public TreeNode left, right;

\* public TreeNode(int val) {

\* this.val = val;

\* this.left = this.right = null;

\* }

\* }

\*/

public class Solution {

/\*\*

\* @param root: the given BST

\* @param target: the given target

\* @return: the value in the BST that is closest to the target

\*/

int closet = Integer.MIN\_VALUE;

public int closestValue(TreeNode root, double target) {

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

stack.push(root);

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

while(root != null) {

stack.push(root);

root = root.left;

}

root = stack.pop();

if(Math.abs(root.val - target) < Math.abs(closet - target)) {

closet = root.val;

}

root = root.right;

}

return closet;

}

}

Time Complexity: O(N)

Space Complexity: O(1)

**Refer to**

<https://wentao-shao.gitbook.io/leetcode/binary-tree/270.closest-binary-search-tree-value>

class Solution {

public int closestValue(TreeNode root, double target) {

LinkedList<TreeNode> stack = new LinkedList();

long pred = Long.MIN\_VALUE;

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

while (root != null) {

stack.add(root);

root = root.left;

}

root = stack.removeLast();

if (pred <= target && target < root.val) {

return Math.abs(pred - target) ? (int)pred : root.val;

}

pred = root.val;

root = root.right;

}

return (int)pred;

}

}