Problem 783: Minimum Distance Between BST Nodes

Problem Information

Difficulty: Easy
Acceptance Rate: 0.00%
Paid Only: No

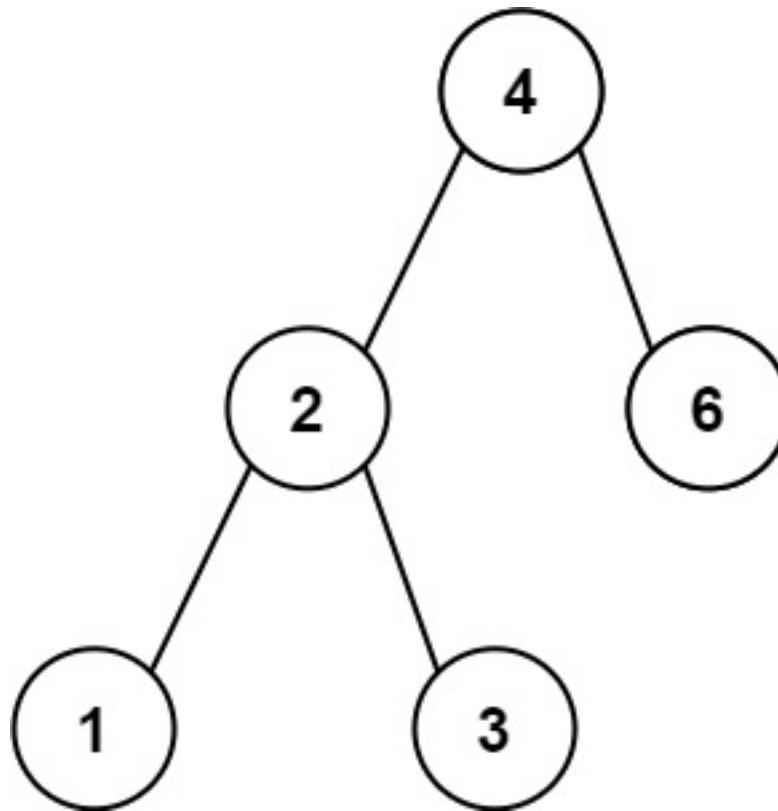## Problem Description

Given the

root

of a Binary Search Tree (BST), return

the minimum difference between the values of any two different nodes in the tree
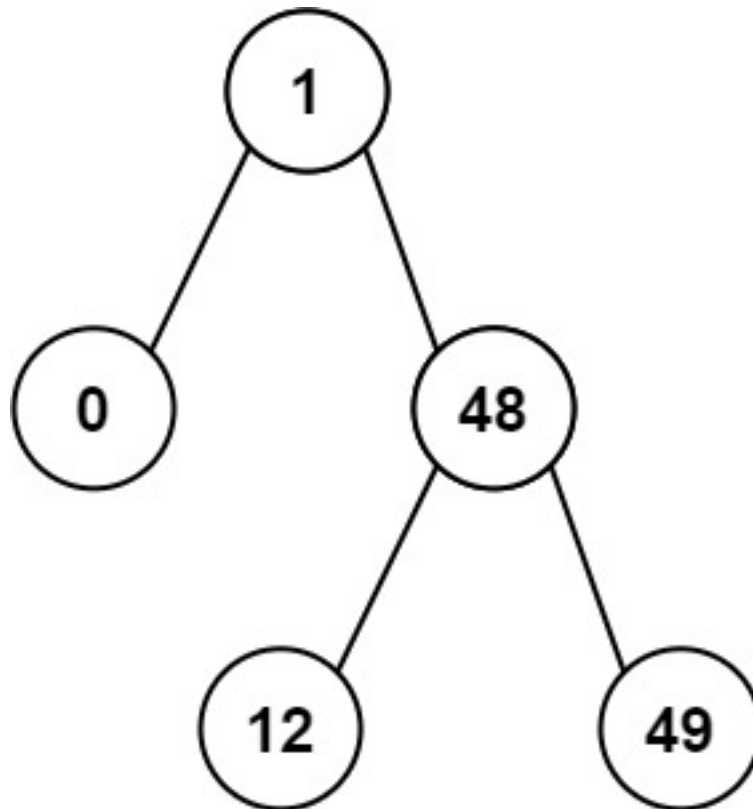
.

Example 1:

Input:

root = [4,2,6,1,3]

Output:

1

Example 2:

Input:

root = [1,0,48,null,null,12,49]

Output:

1

Constraints:

The number of nodes in the tree is in the range

[2, 100]

.

0 <= Node.val <= 10

5

Note:

This question is the same as 530:

https://leetcode.com/problems/minimum-absolute-difference-in-bst/

Code Snippets

C++:

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 * int val;
 * TreeNode *left;
 * TreeNode *right;
 * TreeNode() : val(0), left(nullptr), right(nullptr) {}
 * TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 * TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
 * };
 */
class Solution {
public:
int minDiffInBST(TreeNode* root) {

}
};
```

Java:

```java
/**
 * 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 minDiffInBST(TreeNode root) {

}
}
```

Python3:

```python
# Definition for a binary tree node.
# class TreeNode:
# def __init__(self, val=0, left=None, right=None):
# self.val = val
# self.left = left
# self.right = right
class Solution:
def minDiffInBST(self, root: Optional[TreeNode]) -> int:
```

Python:

```python
# Definition for a binary tree node.
# class TreeNode(object):
# def __init__(self, val=0, left=None, right=None):
# self.val = val
# self.left = left
# self.right = right
class Solution(object):
def minDiffInBST(self, root):
"""
:type root: Optional[TreeNode]
:rtype: int
"""
```

JavaScript:

```javascript
/**
 * Definition for a binary tree node.
 * function TreeNode(val, left, right) {
 * this.val = (val===undefined ? 0 : val)
 * this.left = (left===undefined ? null : left)
```

```
* this.right = (right===undefined ? null : right)
* }
*/
/**
* @param {TreeNode} root
* @return {number}
*/
var minDiffInBST = function(root) {

};
```

TypeScript:

```
/**
* Definition for a binary tree node.
* class TreeNode {
* val: number
* left: TreeNode | null
* right: TreeNode | null
* constructor(val?: number, left?: TreeNode | null, right?: TreeNode | null)
{
* this.val = (val===undefined ? 0 : val)
* this.left = (left===undefined ? null : left)
* this.right = (right===undefined ? null : right)
* }
* }
*/

function minDiffInBST(root: TreeNode | null): number {

};
```

C#:

```
/**
* Definition for a binary tree node.
* public class TreeNode {
* public int val;
* public TreeNode left;
* public TreeNode right;
* public TreeNode(int val=0, TreeNode left=null, TreeNode right=null) {
* this.val = val;
```

```
* this.left = left;
* this.right = right;
* }
* }
*/
public class Solution {
public int MinDiffInBST(TreeNode root) {


}
}
```

C:

```
/**
* Definition for a binary tree node.
* struct TreeNode {
* int val;
* struct TreeNode *left;
* struct TreeNode *right;
* };
*/
int minDiffInBST(struct TreeNode* root) {


}
```

Go:

```
/**
* Definition for a binary tree node.
* type TreeNode struct {
* Val int
* Left *TreeNode
* Right *TreeNode
* }
*/
func minDiffInBST(root *TreeNode) int {


}
```

Kotlin:

```
/**
* Example:
* var ti = TreeNode(5)
* var v = ti.`val`
* Definition for a binary tree node.
* class TreeNode(var `val`: Int) {
* var left: TreeNode? = null
* var right: TreeNode? = null
* }
*/
class Solution {
fun minDiffInBST(root: TreeNode?): Int {


}
}
```

Swift:

```
/**
* Definition for a binary tree node.
* public class TreeNode {
* public var val: Int
* public var left: TreeNode?
* public var right: TreeNode?
* public init() { self.val = 0; self.left = nil; self.right = nil; }
* public init(_ val: Int) { self.val = val; self.left = nil; self.right =
nil; }
* public init(_ val: Int, _ left: TreeNode?, _ right: TreeNode?) {
* self.val = val
* self.left = left
* self.right = right
* }
* }
*/
class Solution {
func minDiffInBST(_ root: TreeNode?) -> Int {


}
}
```

Rust:

```rust
// Definition for a binary tree node.
// #[derive(Debug, PartialEq, Eq)]
// pub struct TreeNode {
// pub val: i32,
// pub left: Option<Rc<RefCell<TreeNode>>>,
// pub right: Option<Rc<RefCell<TreeNode>>>,
// }
//
// impl TreeNode {
// #[inline]
// pub fn new(val: i32) -> Self {
// TreeNode {
// val,
// left: None,
// right: None
// }
// }
// }
use std::rc::Rc;
use std::cell::RefCell;
impl Solution {
pub fn min_diff_in_bst(root: Option<Rc<RefCell<TreeNode>>>) -> i32 {


}
}
```

Ruby:

```ruby
# Definition for a binary tree node.
# class TreeNode
# attr_accessor :val, :left, :right
# def initialize(val = 0, left = nil, right = nil)
# @val = val
# @left = left
# @right = right
# end
# end
# @param {TreeNode} root
# @return {Integer}
def min_diff_in_bst(root)

end
```

PHP:

```php
/**
 * Definition for a binary tree node.
 * class TreeNode {
 * public $val = null;
 * public $left = null;
 * public $right = null;
 * function __construct($val = 0, $left = null, $right = null) {
 * $this->val = $val;
 * $this->left = $left;
 * $this->right = $right;
 * }
 * }
 */
class Solution {

/**
 * @param TreeNode $root
 * @return Integer
 */
function minDiffInBST($root) {

}
}
```

Dart:

```dart
/**
 * Definition for a binary tree node.
 * class TreeNode {
 * int val;
 * TreeNode? left;
 * TreeNode? right;
 * TreeNode([this.val = 0, this.left, this.right]);
 * }
 */
class Solution {
int minDiffInBST(TreeNode? root) {

}
}
```

Scala:

```scala
/**
* Definition for a binary tree node.
* class TreeNode(_value: Int = 0, _left: TreeNode = null, _right: TreeNode =
null) {
* var value: Int = _value
* var left: TreeNode = _left
* var right: TreeNode = _right
* }
*/
object Solution {
def minDiffInBST(root: TreeNode): Int = {


}
}
```

Elixir:

```elixir
# Definition for a binary tree node.
#
# defmodule TreeNode do
# @type t :: %__MODULE__{
# val: integer,
# left: TreeNode.t() | nil,
# right: TreeNode.t() | nil
# }
# defstruct val: 0, left: nil, right: nil
# end

defmodule Solution do
@spec min_diff_in_bst(root :: TreeNode.t | nil) :: integer
def min_diff_in_bst(root) do

end
end
```

Erlang:

```erlang
%% Definition for a binary tree node.
%%
%% -record(tree_node, {val = 0 :: integer(),
%% left = null :: 'null' | #tree_node{},
```

```
%% right = null :: 'null' | #tree_node{}}).


-spec min_diff_in_bst(Root :: #tree_node{} | null) -> integer().

min_diff_in_bst(Root) ->

.
```

Racket:

```
; Definition for a binary tree node.
#|


; val : integer?
; left : (or/c tree-node? #f)
; right : (or/c tree-node? #f)
(struct tree-node
(val left right) #:mutable #:transparent)


; constructor
(define (make-tree-node [val 0])
(tree-node val #f #f))


|#


(define/contract (min-diff-in-bst root)
(-> (or/c tree-node? #f) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Distance Between BST Nodes
* Difficulty: Easy
* Tags: tree, search
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/
```

```
/**
* Definition for a binary tree node.
* struct TreeNode {
* int val;
* TreeNode *left;
* TreeNode *right;
* TreeNode() : val(0), left(nullptr), right(nullptr) {
// TODO: Implement optimized solution
return 0;
}
* TreeNode(int x) : val(x), left(nullptr), right(nullptr) {
// TODO: Implement optimized solution
return 0;
}
* TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {
// TODO: Implement optimized solution
return 0;
}
* };
*/
class Solution {
public:
int minDiffInBST(TreeNode* root) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Distance Between BST Nodes
* Difficulty: Easy
* Tags: tree, search
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/
```

```
/**
* Definition for a binary tree node.
* public class TreeNode {
* int val;
* TreeNode left;
* TreeNode right;
* TreeNode() {
// TODO: Implement optimized solution
return 0;
}
* 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 minDiffInBST(TreeNode root) {


}
}
```

Python3 Solution:

```
"""
Problem: Minimum Distance Between BST Nodes
Difficulty: Easy
Tags: tree, search

Approach: DFS or BFS traversal
Time Complexity: O(n) where n is number of nodes
Space Complexity: O(h) for recursion stack where h is height
"""

# Definition for a binary tree node.
# class TreeNode:
# def __init__(self, val=0, left=None, right=None):
# self.val = val
# self.left = left
```

```python
        # self.right = right
class Solution:
    def minDiffInBST(self, root: Optional[TreeNode]) -> int:
        # TODO: Implement optimized solution
        pass
```

Python Solution:

```python
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution(object):
    def minDiffInBST(self, root):
        """
        :type root: Optional[TreeNode]
        :rtype: int
        """
```

JavaScript Solution:

```javascript
/**
 * Problem: Minimum Distance Between BST Nodes
 * Difficulty: Easy
 * Tags: tree, search
 *
 * Approach: DFS or BFS traversal
 * Time Complexity: O(n) where n is number of nodes
 * Space Complexity: O(h) for recursion stack where h is height
 */


/**
 * Definition for a binary tree node.
 * function TreeNode(val, left, right) {
 *     this.val = (val===undefined ? 0 : val)
 *     this.left = (left===undefined ? null : left)
 *     this.right = (right===undefined ? null : right)
 * }
 */
```

```
/**
 * @param {TreeNode} root
 * @return {number}
 */
var minDiffInBST = function(root) {


};
```

TypeScript Solution:

```typescript
/**
 * Problem: Minimum Distance Between BST Nodes
 * Difficulty: Easy
 * Tags: tree, search
 *
 * Approach: DFS or BFS traversal
 * Time Complexity: O(n) where n is number of nodes
 * Space Complexity: O(h) for recursion stack where h is height
 */


/**
 * Definition for a binary tree node.
 * class TreeNode {
 * val: number
 * left: TreeNode | null
 * right: TreeNode | null
 * constructor(val?: number, left?: TreeNode | null, right?: TreeNode | null)
 * {
 * this.val = (val===undefined ? 0 : val)
 * this.left = (left===undefined ? null : left)
 * this.right = (right===undefined ? null : right)
 * }
 * }
 */


function minDiffInBST(root: TreeNode | null): number {


};
```

C# Solution:

```
/*
* Problem: Minimum Distance Between BST Nodes
* Difficulty: Easy
* Tags: tree, search
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/


/**
* Definition for a binary tree node.
* public class TreeNode {
* public int val;
* public TreeNode left;
* public TreeNode right;
* public TreeNode(int val=0, TreeNode left=null, TreeNode right=null) {
* this.val = val;
* this.left = left;
* this.right = right;
* }
* }
*/
public class Solution {
public int MinDiffInBST(TreeNode root) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Distance Between BST Nodes
* Difficulty: Easy
* Tags: tree, search
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/


/**
```

```
 * Definition for a binary tree node.
 * struct TreeNode {
 * int val;
 * struct TreeNode *left;
 * struct TreeNode *right;
 * };
 */
int minDiffInBST(struct TreeNode* root) {


}
```

Go Solution:

```go
// Problem: Minimum Distance Between BST Nodes
// Difficulty: Easy
// Tags: tree, search
//
// Approach: DFS or BFS traversal
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(h) for recursion stack where h is height

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 * Val int
 * Left *TreeNode
 * Right *TreeNode
 * }
 */
func minDiffInBST(root *TreeNode) int {


}
```

Kotlin Solution:

```kotlin
/**
 * Example:
 * var ti = TreeNode(5)
 * var v = ti.`val`
 * Definition for a binary tree node.
 * class TreeNode(var `val`: Int) {
```

```
* var left: TreeNode? = null
* var right: TreeNode? = null
* }
*/
class Solution {
fun minDiffInBST(root: TreeNode?): Int {


}
}
```

Swift Solution:

```
/**
* Definition for a binary tree node.
* public class TreeNode {
* public var val: Int
* public var left: TreeNode?
* public var right: TreeNode?
* public init() { self.val = 0; self.left = nil; self.right = nil; }
* public init(_ val: Int) { self.val = val; self.left = nil; self.right =
nil; }
* public init(_ val: Int, _ left: TreeNode?, _ right: TreeNode?) {
* self.val = val
* self.left = left
* self.right = right
* }
* }
*/
class Solution {
func minDiffInBST(_ root: TreeNode?) -> Int {


}
}
```

Rust Solution:

```
// Problem: Minimum Distance Between BST Nodes
// Difficulty: Easy
// Tags: tree, search
//
// Approach: DFS or BFS traversal
```

```rust
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(h) for recursion stack where h is height

// Definition for a binary tree node.
// #[derive(Debug, PartialEq, Eq)]
// pub struct TreeNode {
// pub val: i32,
// pub left: Option<Rc<RefCell<TreeNode>>>,
// pub right: Option<Rc<RefCell<TreeNode>>>,
// }
//
// impl TreeNode {
// #[inline]
// pub fn new(val: i32) -> Self {
// TreeNode {
// val,
// left: None,
// right: None
// }
// }
// }
use std::rc::Rc;
use std::cell::RefCell;
impl Solution {
pub fn min_diff_in_bst(root: Option<Rc<RefCell<TreeNode>>>) -> i32 {

}
}
```

Ruby Solution:

```ruby
# Definition for a binary tree node.
# class TreeNode
# attr_accessor :val, :left, :right
# def initialize(val = 0, left = nil, right = nil)
# @val = val
# @left = left
# @right = right
# end
# end
# @param {TreeNode} root
```

```
    # @return {Integer}
    def min_diff_in_bst(root)


    end
```

PHP Solution:

```php
/**
 * Definition for a binary tree node.
 * class TreeNode {
 * public $val = null;
 * public $left = null;
 * public $right = null;
 * function __construct($val = 0, $left = null, $right = null) {
 * $this->val = $val;
 * $this->left = $left;
 * $this->right = $right;
 * }
 * }
 */
class Solution {

    /**
     * @param TreeNode $root
     * @return Integer
     */
    function minDiffInBST($root) {

    }
}
```

Dart Solution:

```dart
/**
 * Definition for a binary tree node.
 * class TreeNode {
 * int val;
 * TreeNode? left;
 * TreeNode? right;
 * TreeNode([this.val = 0, this.left, this.right]);
 * }
```

```
*/
class Solution {
int minDiffInBST(TreeNode? root) {



}
}
```

Scala Solution:

```
/**
* Definition for a binary tree node.
* class TreeNode(_value: Int = 0, _left: TreeNode = null, _right: TreeNode =
null) {
* var value: Int = _value
* var left: TreeNode = _left
* var right: TreeNode = _right
* }
*/
object Solution {
def minDiffInBST(root: TreeNode): Int = {



}
}
```

Elixir Solution:

```
# Definition for a binary tree node.
#
# defmodule TreeNode do
# @type t :: %__MODULE__{
# val: integer,
# left: TreeNode.t() | nil,
# right: TreeNode.t() | nil
# }
# defstruct val: 0, left: nil, right: nil
# end

defmodule Solution do
@spec min_diff_in_bst(root :: TreeNode.t | nil) :: integer
def min_diff_in_bst(root) do
```

```
    end
end
```

Erlang Solution:

```erlang
%% Definition for a binary tree node.
%%
%% -record(tree_node, {val = 0 :: integer(),
%% left = null :: 'null' | #tree_node{},
%% right = null :: 'null' | #tree_node{}}).


-spec min_diff_in_bst(Root :: #tree_node{} | null) -> integer().
min_diff_in_bst(Root) ->

  .
```

Racket Solution:

```racket
; Definition for a binary tree node.
#|

; val : integer?
; left : (or/c tree-node? #f)
; right : (or/c tree-node? #f)
(struct tree-node
(val left right) #:mutable #:transparent)

; constructor
(define (make-tree-node [val 0])
(tree-node val #f #f))


|#

(define/contract (min-diff-in-bst root)
(-> (or/c tree-node? #f) exact-integer?)
)
```