Problem 173: Binary Search Tree Iterator

Problem Information

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

Problem Description

Implement the

BSTIterator

class that represents an iterator over the

in-order traversal

of a binary search tree (BST):

BSTIterator(TreeNode root)

Initializes an object of the

BSTIterator

class. The

root

of the BST is given as part of the constructor. The pointer should be initialized to a non-existent number smaller than any element in the BST.

boolean hasNext()

Returns

true

if there exists a number in the traversal to the right of the pointer, otherwise returns

false

.

int next()

Moves the pointer to the right, then returns the number at the pointer.

Notice that by initializing the pointer to a non-existent smallest number, the first call to

next()

will return the smallest element in the BST.

You may assume that

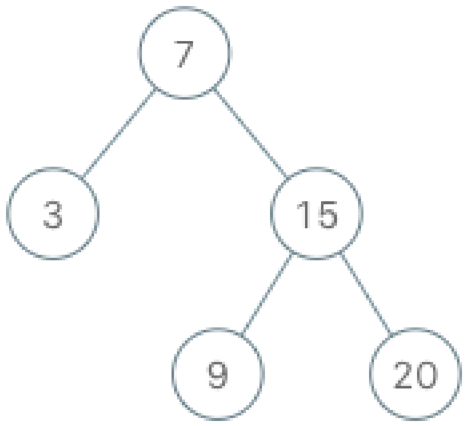
next()

calls will always be valid. That is, there will be at least a next number in the in-order traversal when

next()

is called.

Example 1:



Input

["BSTIterator", "next", "next", "hasNext", "next", "hasNext", "next", "hasNext", "next", "hasNext"] [[[7, 3, 15, null, null, 9, 20]], [], [], [], [], [], [], [], [], []]

Output

[null, 3, 7, true, 9, true, 15, true, 20, false]

Explanation

BSTIterator bSTIterator = new BSTIterator([7, 3, 15, null, null, 9, 20]); bSTIterator.next(); // return 3 bSTIterator.next(); // return 7 bSTIterator.hasNext(); // return True bSTIterator.next(); // return 9 bSTIterator.hasNext(); // return True bSTIterator.next(); // return 15 bSTIterator.hasNext(); // return True bSTIterator.next(); // return 20 bSTIterator.hasNext(); // return False

Constraints:

The number of nodes in the tree is in the range

[1, 10

5

]

.

0 <= Node.val <= 10

6

At most

10

5

calls will be made to

hasNext

, and

next

.

Follow up:

Could you implement

next()

and

hasNext()

to run in average

O(1)

time and use

O(h)

memory, where

h

is the height of the tree?

Code Snippets

C++:

```
/**
* 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 BSTIterator {
public:
BSTIterator(TreeNode* root) {

}

int next() {

}

bool hasNext() {

}
};

/**
* Your BSTIterator object will be instantiated and called as such:
* BSTIterator* obj = new BSTIterator(root);
* int param_1 = obj->next();
* bool param_2 = obj->hasNext();
*/
```

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 BSTIterator {

public BSTIterator(TreeNode root) {

}

public int next() {

}

public boolean hasNext() {

}
}

/**
* Your BSTIterator object will be instantiated and called as such:
* BSTIterator obj = new BSTIterator(root);
* int param_1 = obj.next();
* boolean param_2 = obj.hasNext();
*/
```

Python3:

```
# 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 BSTIterator:

def __init__(self, root: Optional[TreeNode]):



def next(self) -> int:



def hasNext(self) -> bool:




# Your BSTIterator object will be instantiated and called as such:
# obj = BSTIterator(root)
# param_1 = obj.next()
# param_2 = obj.hasNext()
```

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 BSTIterator(object):

def __init__(self, root):
"""
:type root: Optional[TreeNode]
"""



def next(self):
```

```
"""
:rtype: int
"""



def hasNext(self):
"""
:rtype: bool
"""




# Your BSTIterator object will be instantiated and called as such:
# obj = BSTIterator(root)
# param_1 = obj.next()
# param_2 = obj.hasNext()
```

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
*/
var BSTIterator = function(root) {

};

/**
* @return {number}
*/
BSTIterator.prototype.next = function() {

};

/**
```

```
* @return {boolean}
*/
BSTIterator.prototype.hasNext = function() {

};

/**
* Your BSTIterator object will be instantiated and called as such:
* var obj = new BSTIterator(root)
* var param_1 = obj.next()
* var param_2 = obj.hasNext()
*/
```

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)
* }
* }
*/

class BSTIterator {
constructor(root: TreeNode | null) {

}

next(): number {

}

hasNext(): boolean {

}
}

/**
* Your BSTIterator object will be instantiated and called as such:
* var obj = new BSTIterator(root)
* var param_1 = obj.next()
* var param_2 = obj.hasNext()
*/
```

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 BSTIterator {

public BSTIterator(TreeNode root) {

}

public int Next() {

}

public bool HasNext() {

}
}

/**
* Your BSTIterator object will be instantiated and called as such:
* BSTIterator obj = new BSTIterator(root);
* int param_1 = obj.Next();
* bool param_2 = obj.HasNext();
*/
```

C:

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



typedef struct {

} BSTIterator;


BSTIterator* bSTIteratorCreate(struct TreeNode* root) {

}

int bSTIteratorNext(BSTIterator* obj) {

}

bool bSTIteratorHasNext(BSTIterator* obj) {

}

void bSTIteratorFree(BSTIterator* obj) {

}

/**
* Your BSTIterator struct will be instantiated and called as such:
* BSTIterator* obj = bSTIteratorCreate(root);
* int param_1 = bSTIteratorNext(obj);
* bool param_2 = bSTIteratorHasNext(obj);

* bSTIteratorFree(obj);
*/
```

Go:

```
/**
* Definition for a binary tree node.
* type TreeNode struct {
* Val int
* Left *TreeNode
* Right *TreeNode
* }
*/
type BSTIterator struct {

}



func Constructor(root *TreeNode) BSTIterator {

}



func (this *BSTIterator) Next() int {

}



func (this *BSTIterator) HasNext() bool {

}



/**
* Your BSTIterator object will be instantiated and called as such:
* obj := Constructor(root);
* param_1 := obj.Next();
* param_2 := obj.HasNext();
*/
```

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 BSTIterator(root: TreeNode?) {

fun next(): Int {

}

fun hasNext(): Boolean {

}

}

/**
* Your BSTIterator object will be instantiated and called as such:
* var obj = BSTIterator(root)
* var param_1 = obj.next()
* var param_2 = obj.hasNext()
*/
```

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 BSTIterator {

init(_ root: TreeNode?) {

}

func next() -> Int {

}

func hasNext() -> Bool {

}
}

/**
* Your BSTIterator object will be instantiated and called as such:
* let obj = BSTIterator(root)
* let ret_1: Int = obj.next()
* let ret_2: Bool = obj.hasNext()
*/
```

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
// }
// }
// }
struct BSTIterator {

}



/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl BSTIterator {

fn new(root: Option<Rc<RefCell<TreeNode>>>) -> Self {

}

fn next(&self) -> i32 {

}

fn has_next(&self) -> bool {

}
}

/**
* Your BSTIterator object will be instantiated and called as such:
* let obj = BSTIterator::new(root);
* let ret_1: i32 = obj.next();
* let ret_2: bool = obj.has_next();
*/
```

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
class BSTIterator

=begin
:type root: TreeNode
=end
def initialize(root)

end



=begin
:rtype: Integer
=end
def next()

end



=begin
:rtype: Boolean
=end
def has_next()

end



end

# Your BSTIterator object will be instantiated and called as such:
# obj = BSTIterator.new(root)
# param_1 = obj.next()
# param_2 = obj.has_next()
```

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 BSTIterator {
/**
* @param TreeNode $root
*/
function __construct($root) {

}

/**
* @return Integer
*/
function next() {

}

/**
* @return Boolean
*/
function hasNext() {

}
}

/**
* Your BSTIterator object will be instantiated and called as such:
* $obj = BSTIterator($root);
* $ret_1 = $obj->next();
* $ret_2 = $obj->hasNext();
```

```
*/
```

Dart:

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

BSTIterator(TreeNode? root) {

}

int next() {

}

bool hasNext() {

}
}

/**
* Your BSTIterator object will be instantiated and called as such:
* BSTIterator obj = BSTIterator(root);
* int param1 = obj.next();
* bool param2 = obj.hasNext();
*/
```

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
* }
*/
class BSTIterator(_root: TreeNode) {

def next(): Int = {

}

def hasNext(): Boolean = {

}

}

/**
* Your BSTIterator object will be instantiated and called as such:
* val obj = new BSTIterator(root)
* val param_1 = obj.next()
* val param_2 = obj.hasNext()
*/
```

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 BSTIterator do
@spec init_(root :: TreeNode.t | nil) :: any
def init_(root) do

end
```

```
@spec next() :: integer
def next() do

end

@spec has_next() :: boolean
def has_next() do

end
end

# Your functions will be called as such:
# BSTIterator.init_(root)
# param_1 = BSTIterator.next()
# param_2 = BSTIterator.has_next()

# BSTIterator.init_ will be called before every test case, in which you can
do some necessary initializations.
```

Erlang:

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

-spec bst_iterator_init_(Root :: #tree_node{} | null) -> any().
bst_iterator_init_(Root) ->
.

-spec bst_iterator_next() -> integer().
bst_iterator_next() ->
.

-spec bst_iterator_has_next() -> boolean().
bst_iterator_has_next() ->
.



%% Your functions will be called as such:
%% bst_iterator_init_(Root),
%% Param_1 = bst_iterator_next(),
%% Param_2 = bst_iterator_has_next(),

%% bst_iterator_init_ will be called before every test case, in which you can
do some necessary initializations.
```

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 bst-iterator%
(class object%
(super-new)

; root : (or/c tree-node? #f)
(init-field
root)

; next : -> exact-integer?
(define/public (next)
)
; has-next : -> boolean?
(define/public (has-next)
)))

;; Your bst-iterator% object will be instantiated and called as such:
;; (define obj (new bst-iterator% [root root]))
;; (define param_1 (send obj next))
;; (define param_2 (send obj has-next))
```

Solutions

C++ Solution:

```
/*
* Problem: Binary Search Tree Iterator
* Difficulty: Medium
* Tags: tree, search, stack
*
* 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 BSTIterator {
public:
BSTIterator(TreeNode* root) {

}
int next() {

}

bool hasNext() {

}
};

/**
* Your BSTIterator object will be instantiated and called as such:
* BSTIterator* obj = new BSTIterator(root);
* int param_1 = obj->next();
* bool param_2 = obj->hasNext();
*/
```

Java Solution:

```
/**
* Problem: Binary Search Tree Iterator
* Difficulty: Medium
* Tags: tree, search, stack
*
* 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 BSTIterator {

public BSTIterator(TreeNode root) {

}

public int next() {

}

public boolean hasNext() {

}
}

/**
* Your BSTIterator object will be instantiated and called as such:
* BSTIterator obj = new BSTIterator(root);
* int param_1 = obj.next();
* boolean param_2 = obj.hasNext();
*/
```

Python3 Solution:

```
"""
Problem: Binary Search Tree Iterator
Difficulty: Medium
Tags: tree, search, stack

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
# self.right = right
class BSTIterator:

def __init__(self, root: Optional[TreeNode]):



def next(self) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
# 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 BSTIterator(object):

def __init__(self, root):
"""
:type root: Optional[TreeNode]
"""



def next(self):
"""
:rtype: int
"""



def hasNext(self):
"""
:rtype: bool
"""
```

```
# Your BSTIterator object will be instantiated and called as such:
# obj = BSTIterator(root)
# param_1 = obj.next()
# param_2 = obj.hasNext()
```

JavaScript Solution:

```
/**
* Problem: Binary Search Tree Iterator
* Difficulty: Medium
* Tags: tree, search, stack
*
* 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
*/
var BSTIterator = function(root) {

};

/**
* @return {number}
*/
BSTIterator.prototype.next = function() {

};

/**
```

```
* @return {boolean}
*/
BSTIterator.prototype.hasNext = function() {

};

/**
* Your BSTIterator object will be instantiated and called as such:
* var obj = new BSTIterator(root)
* var param_1 = obj.next()
* var param_2 = obj.hasNext()
*/
```

TypeScript Solution:

```
/**
* Problem: Binary Search Tree Iterator
* Difficulty: Medium
* Tags: tree, search, stack
*
* 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)
* }
* }
*/

class BSTIterator {
```

```
constructor(root: TreeNode | null) {

}

next(): number {

}

hasNext(): boolean {

}
}

/**
* Your BSTIterator object will be instantiated and called as such:
* var obj = new BSTIterator(root)
* var param_1 = obj.next()
* var param_2 = obj.hasNext()
*/
```

C# Solution:

```
/*
* Problem: Binary Search Tree Iterator
* Difficulty: Medium
* Tags: tree, search, stack
*
* 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 BSTIterator {

public BSTIterator(TreeNode root) {

}

public int Next() {

}

public bool HasNext() {

}
}

/**
* Your BSTIterator object will be instantiated and called as such:
* BSTIterator obj = new BSTIterator(root);
* int param_1 = obj.Next();
* bool param_2 = obj.HasNext();
*/
```

C Solution:

```
/*
* Problem: Binary Search Tree Iterator
* Difficulty: Medium
* Tags: tree, search, stack
*
* 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;
* };
*/



typedef struct {

} BSTIterator;


BSTIterator* bSTIteratorCreate(struct TreeNode* root) {

}

int bSTIteratorNext(BSTIterator* obj) {

}

bool bSTIteratorHasNext(BSTIterator* obj) {

}

void bSTIteratorFree(BSTIterator* obj) {

}

/**
* Your BSTIterator struct will be instantiated and called as such:
* BSTIterator* obj = bSTIteratorCreate(root);
* int param_1 = bSTIteratorNext(obj);

* bool param_2 = bSTIteratorHasNext(obj);

* bSTIteratorFree(obj);
*/
```

Go Solution:

```
// Problem: Binary Search Tree Iterator
// Difficulty: Medium
// Tags: tree, search, stack
//
// 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
* }
*/
type BSTIterator struct {

}



func Constructor(root *TreeNode) BSTIterator {

}



func (this *BSTIterator) Next() int {

}



func (this *BSTIterator) HasNext() bool {

}



/**
* Your BSTIterator object will be instantiated and called as such:
* obj := Constructor(root);
* param_1 := obj.Next();
* param_2 := obj.HasNext();
*/
```

Kotlin Solution:

```
/**
* 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 BSTIterator(root: TreeNode?) {

fun next(): Int {

}

fun hasNext(): Boolean {

}

}

/**
* Your BSTIterator object will be instantiated and called as such:
* var obj = BSTIterator(root)
* var param_1 = obj.next()
* var param_2 = obj.hasNext()
*/
```

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 BSTIterator {

init(_ root: TreeNode?) {

}

func next() -> Int {

}

func hasNext() -> Bool {

}
}

/**
* Your BSTIterator object will be instantiated and called as such:
* let obj = BSTIterator(root)
* let ret_1: Int = obj.next()
* let ret_2: Bool = obj.hasNext()
*/
```

Rust Solution:

```
// Problem: Binary Search Tree Iterator
// Difficulty: Medium
// Tags: tree, search, stack
//
// 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.
// #[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
// }
// }
// }
struct BSTIterator {

}



/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl BSTIterator {

fn new(root: Option<Rc<RefCell<TreeNode>>>) -> Self {

}

fn next(&self) -> i32 {

}

fn has_next(&self) -> bool {

}
}

/**
```

```
* Your BSTIterator object will be instantiated and called as such:
* let obj = BSTIterator::new(root);
* let ret_1: i32 = obj.next();
* let ret_2: bool = obj.has_next();
*/
```

Ruby Solution:

```
# 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
class BSTIterator

=begin
:type root: TreeNode
=end
def initialize(root)

end



=begin
:rtype: Integer
=end
def next()

end



=begin
:rtype: Boolean
=end
def has_next()

end
```

```



end

# Your BSTIterator object will be instantiated and called as such:
# obj = BSTIterator.new(root)
# param_1 = obj.next()
# param_2 = obj.has_next()
```

PHP Solution:

```
/**
* 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 BSTIterator {
/**
* @param TreeNode $root
*/
function __construct($root) {

}

/**
* @return Integer
*/
function next() {

}

/**
* @return Boolean
*/
function hasNext() {

}
}

/**
* Your BSTIterator object will be instantiated and called as such:
* $obj = BSTIterator($root);
* $ret_1 = $obj->next();
* $ret_2 = $obj->hasNext();
*/
```

Dart Solution:

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

BSTIterator(TreeNode? root) {

}

int next() {

}

bool hasNext() {

}
}

/**
* Your BSTIterator object will be instantiated and called as such:
```

```
* BSTIterator obj = BSTIterator(root);
* int param1 = obj.next();
* bool param2 = obj.hasNext();
*/
```

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
* }
*/
class BSTIterator(_root: TreeNode) {

def next(): Int = {

}

def hasNext(): Boolean = {

}

}

/**
* Your BSTIterator object will be instantiated and called as such:
* val obj = new BSTIterator(root)
* val param_1 = obj.next()
* val param_2 = obj.hasNext()
*/
```

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 BSTIterator do
@spec init_(root :: TreeNode.t | nil) :: any
def init_(root) do

end

@spec next() :: integer
def next() do

end

@spec has_next() :: boolean
def has_next() do

end
end

# Your functions will be called as such:
# BSTIterator.init_(root)
# param_1 = BSTIterator.next()
# param_2 = BSTIterator.has_next()

# BSTIterator.init_ will be called before every test case, in which you can
do some necessary initializations.
```

Erlang Solution:

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

-spec bst_iterator_init_(Root :: #tree_node{} | null) -> any().
```

```
bst_iterator_init_(Root) ->
.

-spec bst_iterator_next() -> integer().
bst_iterator_next() ->
.

-spec bst_iterator_has_next() -> boolean().
bst_iterator_has_next() ->
.



%% Your functions will be called as such:
%% bst_iterator_init_(Root),
%% Param_1 = bst_iterator_next(),
%% Param_2 = bst_iterator_has_next(),

%% bst_iterator_init_ will be called before every test case, in which you can
do some necessary initializations.
```

Racket Solution:

```
; 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 bst-iterator%
(class object%
(super-new)
; root : (or/c tree-node? #f)
(init-field
root)

; next : -> exact-integer?
(define/public (next)
)
; has-next : -> boolean?
(define/public (has-next)
)))

;; Your bst-iterator% object will be instantiated and called as such:
;; (define obj (new bst-iterator% [root root]))
;; (define param_1 (send obj next))
;; (define param_2 (send obj has-next))
```