Problem 919: Complete Binary Tree Inserter

Problem Information

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

Problem Description

A

complete binary tree

is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible.

Design an algorithm to insert a new node to a complete binary tree keeping it complete after the insertion.

Implement the

CBTInserter

class:

CBTInserter(TreeNode root)

Initializes the data structure with the

root

of the complete binary tree.

int insert(int v)

Inserts a

TreeNode

into the tree with value

Node.val == val

so that the tree remains complete, and returns the value of the parent of the inserted

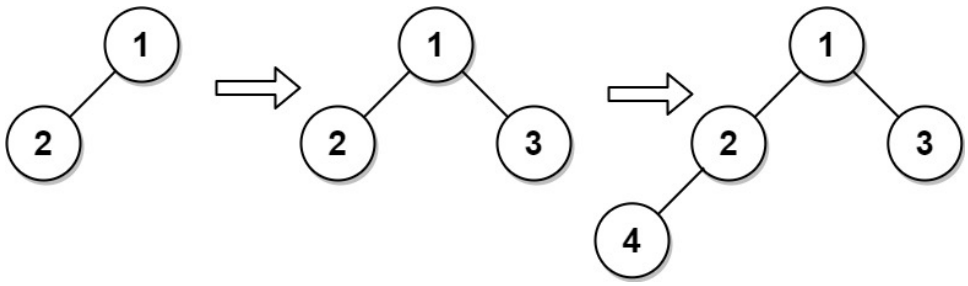
TreeNode

.

TreeNode get_root()

Returns the root node of the tree.

Example 1:



Input

["CBTInserter", "insert", "insert", "get_root"] [[[1, 2]], [3], [4], []]

Output

[null, 1, 2, [1, 2, 3, 4]]

Explanation

CBTInserter cBTInserter = new CBTInserter([1, 2]); cBTInserter.insert(3); // return 1
cBTInserter.insert(4); // return 2 cBTInserter.get_root(); // return [1, 2, 3, 4]

Constraints:

The number of nodes in the tree will be in the range

[1, 1000]

.

0 <= Node.val <= 5000

root

is a complete binary tree.

0 <= val <= 5000

At most

10

4

calls will be made to

insert

and

get_root

.

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 CBTInserter {
public:
CBTInserter(TreeNode* root) {

}

int insert(int val) {

}

TreeNode* get_root() {

}
};

/**
* Your CBTInserter object will be instantiated and called as such:
* CBTInserter* obj = new CBTInserter(root);
* int param_1 = obj->insert(val);
* TreeNode* param_2 = obj->get_root();
*/
```

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 CBTInserter {

public CBTInserter(TreeNode root) {

}

public int insert(int val) {

}

public TreeNode get_root() {

}
}

/**
* Your CBTInserter object will be instantiated and called as such:
* CBTInserter obj = new CBTInserter(root);
* int param_1 = obj.insert(val);
* TreeNode param_2 = obj.get_root();
*/
```

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 CBTInserter:

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

```
def insert(self, val: int) -> int:


def get_root(self) -> Optional[TreeNode]:



# Your CBTInserter object will be instantiated and called as such:
# obj = CBTInserter(root)
# param_1 = obj.insert(val)
# param_2 = obj.get_root()
```

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 CBTInserter(object):

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



def insert(self, val):
"""
:type val: int
:rtype: int
"""



def get_root(self):
"""
:rtype: Optional[TreeNode]
"""
```

```
# Your CBTInserter object will be instantiated and called as such:
# obj = CBTInserter(root)
# param_1 = obj.insert(val)
# param_2 = obj.get_root()
```

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 CBTInserter = function(root) {

};

/**
* @param {number} val
* @return {number}
*/
CBTInserter.prototype.insert = function(val) {

};

/**
* @return {TreeNode}
*/
CBTInserter.prototype.get_root = function() {

};

/**
* Your CBTInserter object will be instantiated and called as such:
* var obj = new CBTInserter(root)
* var param_1 = obj.insert(val)
* var param_2 = obj.get_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)
* }
* }
*/

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

}

insert(val: number): number {

}

get_root(): TreeNode | null {

}
}

/**
* Your CBTInserter object will be instantiated and called as such:
* var obj = new CBTInserter(root)
* var param_1 = obj.insert(val)
* var param_2 = obj.get_root()
*/
```

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 CBTInserter {

public CBTInserter(TreeNode root) {

}

public int Insert(int val) {

}

public TreeNode Get_root() {

}
}

/**
* Your CBTInserter object will be instantiated and called as such:
* CBTInserter obj = new CBTInserter(root);
* int param_1 = obj.Insert(val);
* TreeNode param_2 = obj.Get_root();
*/
```

C:

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

```
* };
*/



typedef struct {

} CBTInserter;


CBTInserter* cBTInserterCreate(struct TreeNode* root) {

}

int cBTInserterInsert(CBTInserter* obj, int val) {

}

struct TreeNode* cBTInserterGet_root(CBTInserter* obj) {

}

void cBTInserterFree(CBTInserter* obj) {

}

/**
* Your CBTInserter struct will be instantiated and called as such:
* CBTInserter* obj = cBTInserterCreate(root);
* int param_1 = cBTInserterInsert(obj, val);

* struct TreeNode* param_2 = cBTInserterGet_root(obj);

* cBTInserterFree(obj);
*/
```

Go:

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

```
* Left *TreeNode
* Right *TreeNode
* }
*/
type CBTInserter struct {

}


func Constructor(root *TreeNode) CBTInserter {

}


func (this *CBTInserter) Insert(val int) int {

}


func (this *CBTInserter) Get_root() *TreeNode {

}


/**
* Your CBTInserter object will be instantiated and called as such:
* obj := Constructor(root);
* param_1 := obj.Insert(val);
* param_2 := obj.Get_root();
*/
```

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 CBTInserter(root: TreeNode?) {

fun insert(`val`: Int): Int {

}

fun get_root(): TreeNode? {

}

}

/**
* Your CBTInserter object will be instantiated and called as such:
* var obj = CBTInserter(root)
* var param_1 = obj.insert(`val`)
* var param_2 = obj.get_root()
*/
```

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 CBTInserter {

init(_ root: TreeNode?) {
```

```
}

func insert(_ val: Int) -> Int {

}

func get_root() -> TreeNode? {

}
}

/**
* Your CBTInserter object will be instantiated and called as such:
* let obj = CBTInserter(root)
* let ret_1: Int = obj.insert(val)
* let ret_2: TreeNode? = obj.get_root()
*/
```

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 CBTInserter {

}
```

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

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

}

fn insert(&self, val: i32) -> i32 {

}

fn get_root(&self) -> Option<Rc<RefCell<TreeNode>>> {

}
}

/**
* Your CBTInserter object will be instantiated and called as such:
* let obj = CBTInserter::new(root);
* let ret_1: i32 = obj.insert(val);
* let ret_2: Option<Rc<RefCell<TreeNode>>> = obj.get_root();
*/
```

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 CBTInserter

=begin
```

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

end


=begin
:type val: Integer
:rtype: Integer
=end
def insert(val)

end


=begin
:rtype: TreeNode
=end
def get_root()

end


end

# Your CBTInserter object will be instantiated and called as such:
# obj = CBTInserter.new(root)
# param_1 = obj.insert(val)
# param_2 = obj.get_root()
```

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

}

/**
* @param Integer $val
* @return Integer
*/
function insert($val) {

}

/**
* @return TreeNode
*/
function get_root() {

}
}

/**
* Your CBTInserter object will be instantiated and called as such:
* $obj = CBTInserter($root);
* $ret_1 = $obj->insert($val);
* $ret_2 = $obj->get_root();
*/
```

Dart:

```
/**
* Definition for a binary tree node.
* class TreeNode {
* int val;
* TreeNode? left;
```

```
* TreeNode? right;
* TreeNode([this.val = 0, this.left, this.right]);
* }
*/
class CBTInserter {

CBTInserter(TreeNode? root) {

}

int insert(int val) {

}

TreeNode? get_root() {

}
}

/**
* Your CBTInserter object will be instantiated and called as such:
* CBTInserter obj = CBTInserter(root);
* int param1 = obj.insert(val);
* TreeNode? param2 = obj.get_root();
*/
```

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 CBTInserter(_root: TreeNode) {

def insert(`val`: Int): Int = {

}
```

```
def get_root(): TreeNode = {

}

}

/**
* Your CBTInserter object will be instantiated and called as such:
* val obj = new CBTInserter(root)
* val param_1 = obj.insert(`val`)
* val param_2 = obj.get_root()
*/
```

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

end

@spec insert(val :: integer) :: integer
def insert(val) do

end

@spec get_root() :: TreeNode.t | nil
def get_root() do

end
```

```
end

# Your functions will be called as such:
# CBTInserter.init_(root)
# param_1 = CBTInserter.insert(val)
# param_2 = CBTInserter.get_root()

# CBTInserter.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 cbt_inserter_init_(Root :: #tree_node{} | null) -> any().
cbt_inserter_init_(Root) ->
.

-spec cbt_inserter_insert(Val :: integer()) -> integer().
cbt_inserter_insert(Val) ->
.

-spec cbt_inserter_get_root() -> #tree_node{} | null.
cbt_inserter_get_root() ->
.


%% Your functions will be called as such:
%% cbt_inserter_init_(Root),
%% Param_1 = cbt_inserter_insert(Val),
%% Param_2 = cbt_inserter_get_root(),

%% cbt_inserter_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 cbt-inserter%
(class object%
(super-new)

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

; insert : exact-integer? -> exact-integer?
(define/public (insert val)
)
; get_root : -> (or/c tree-node? #f)
(define/public (get_root)
)))

;; Your cbt-inserter% object will be instantiated and called as such:
;; (define obj (new cbt-inserter% [root root]))
;; (define param_1 (send obj insert val))
;; (define param_2 (send obj get_root))
```

Solutions

C++ Solution:

```
/*
* Problem: Complete Binary Tree Inserter
```

```
* Difficulty: Medium
* 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 CBTInserter {
public:
CBTInserter(TreeNode* root) {

}

int insert(int val) {

}

TreeNode* get_root() {

}
```

```
};

/**
* Your CBTInserter object will be instantiated and called as such:
* CBTInserter* obj = new CBTInserter(root);
* int param_1 = obj->insert(val);
* TreeNode* param_2 = obj->get_root();
*/
```

Java Solution:

```
/**
* Problem: Complete Binary Tree Inserter
* Difficulty: Medium
* 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 CBTInserter {
```

```
public CBTInserter(TreeNode root) {

}

public int insert(int val) {

}

public TreeNode get_root() {

}
}

/**
* Your CBTInserter object will be instantiated and called as such:
* CBTInserter obj = new CBTInserter(root);
* int param_1 = obj.insert(val);
* TreeNode param_2 = obj.get_root();
*/
```

Python3 Solution:

```
"""
Problem: Complete Binary Tree Inserter
Difficulty: Medium
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
# self.right = right
class CBTInserter:

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

```
def insert(self, val: int) -> 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 CBTInserter(object):

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


def insert(self, val):
"""
:type val: int
:rtype: int
"""


def get_root(self):
"""
:rtype: Optional[TreeNode]
"""



# Your CBTInserter object will be instantiated and called as such:
# obj = CBTInserter(root)
# param_1 = obj.insert(val)
# param_2 = obj.get_root()
```

JavaScript Solution:

```
/**
* Problem: Complete Binary Tree Inserter
* Difficulty: Medium
* 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
*/
var CBTInserter = function(root) {

};

/**
* @param {number} val
* @return {number}
*/
CBTInserter.prototype.insert = function(val) {

};

/**
* @return {TreeNode}
*/
CBTInserter.prototype.get_root = function() {

};

/**
```

```
* Your CBTInserter object will be instantiated and called as such:
* var obj = new CBTInserter(root)
* var param_1 = obj.insert(val)
* var param_2 = obj.get_root()
*/
```

TypeScript Solution:

```
/**
* Problem: Complete Binary Tree Inserter
* Difficulty: Medium
* 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)
* }
* }
*/

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

}

insert(val: number): number {

}
```

```
get_root(): TreeNode | null {

}
}

/**
* Your CBTInserter object will be instantiated and called as such:
* var obj = new CBTInserter(root)
* var param_1 = obj.insert(val)
* var param_2 = obj.get_root()
*/
```

C# Solution:

```
/*
* Problem: Complete Binary Tree Inserter
* Difficulty: Medium
* 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 CBTInserter {

public CBTInserter(TreeNode root) {
```

```
}

public int Insert(int val) {

}

public TreeNode Get_root() {

}
}

/**
* Your CBTInserter object will be instantiated and called as such:
* CBTInserter obj = new CBTInserter(root);
* int param_1 = obj.Insert(val);
* TreeNode param_2 = obj.Get_root();
*/
```

C Solution:

```
/*
* Problem: Complete Binary Tree Inserter
* Difficulty: Medium
* 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;
* };
*/
```

```
typedef struct {

} CBTInserter;


CBTInserter* cBTInserterCreate(struct TreeNode* root) {

}

int cBTInserterInsert(CBTInserter* obj, int val) {

}

struct TreeNode* cBTInserterGet_root(CBTInserter* obj) {

}

void cBTInserterFree(CBTInserter* obj) {

}

/**
* Your CBTInserter struct will be instantiated and called as such:
* CBTInserter* obj = cBTInserterCreate(root);
* int param_1 = cBTInserterInsert(obj, val);

* struct TreeNode* param_2 = cBTInserterGet_root(obj);

* cBTInserterFree(obj);
*/
```

Go Solution:

```
// Problem: Complete Binary Tree Inserter
// Difficulty: Medium
// 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
* }
*/
type CBTInserter struct {

}


func Constructor(root *TreeNode) CBTInserter {

}


func (this *CBTInserter) Insert(val int) int {

}


func (this *CBTInserter) Get_root() *TreeNode {

}


/**
* Your CBTInserter object will be instantiated and called as such:
* obj := Constructor(root);
* param_1 := obj.Insert(val);
* param_2 := obj.Get_root();
*/
```

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 CBTInserter(root: TreeNode?) {

fun insert(`val`: Int): Int {

}

fun get_root(): TreeNode? {

}

}

/**
* Your CBTInserter object will be instantiated and called as such:
* var obj = CBTInserter(root)
* var param_1 = obj.insert(`val`)
* var param_2 = obj.get_root()
*/
```

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 CBTInserter {

init(_ root: TreeNode?) {

}

func insert(_ val: Int) -> Int {

}

func get_root() -> TreeNode? {

}
}

/**
* Your CBTInserter object will be instantiated and called as such:
* let obj = CBTInserter(root)
* let ret_1: Int = obj.insert(val)
* let ret_2: TreeNode? = obj.get_root()
*/
```

Rust Solution:

```
// Problem: Complete Binary Tree Inserter
// Difficulty: Medium
// 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.
// #[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 CBTInserter {

}


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

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

}

fn insert(&self, val: i32) -> i32 {

}

fn get_root(&self) -> Option<Rc<RefCell<TreeNode>>> {

}
}

/**
* Your CBTInserter object will be instantiated and called as such:
* let obj = CBTInserter::new(root);
* let ret_1: i32 = obj.insert(val);
* let ret_2: Option<Rc<RefCell<TreeNode>>> = obj.get_root();
```

```
*/
```

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 CBTInserter

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

end


=begin
:type val: Integer
:rtype: Integer
=end
def insert(val)

end


=begin
:rtype: TreeNode
=end
def get_root()

end


end
```

```
# Your CBTInserter object will be instantiated and called as such:
# obj = CBTInserter.new(root)
# param_1 = obj.insert(val)
# param_2 = obj.get_root()
```

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

}

/**
* @param Integer $val
* @return Integer
*/
function insert($val) {

}

/**
* @return TreeNode
*/
function get_root() {
```

```
}
}

/**
* Your CBTInserter object will be instantiated and called as such:
* $obj = CBTInserter($root);
* $ret_1 = $obj->insert($val);
* $ret_2 = $obj->get_root();
*/
```

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 CBTInserter {

CBTInserter(TreeNode? root) {

}

int insert(int val) {

}

TreeNode? get_root() {

}
}

/**
* Your CBTInserter object will be instantiated and called as such:
* CBTInserter obj = CBTInserter(root);
* int param1 = obj.insert(val);
```

```
* TreeNode? param2 = obj.get_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
* }
*/
class CBTInserter(_root: TreeNode) {

def insert(`val`: Int): Int = {

}

def get_root(): TreeNode = {

}

}

/**
* Your CBTInserter object will be instantiated and called as such:
* val obj = new CBTInserter(root)
* val param_1 = obj.insert(`val`)
* val param_2 = obj.get_root()
*/
```

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

end

@spec insert(val :: integer) :: integer
def insert(val) do

end

@spec get_root() :: TreeNode.t | nil
def get_root() do

end
end

# Your functions will be called as such:
# CBTInserter.init_(root)
# param_1 = CBTInserter.insert(val)
# param_2 = CBTInserter.get_root()

# CBTInserter.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 cbt_inserter_init_(Root :: #tree_node{} | null) -> any().
cbt_inserter_init_(Root) ->
.
```

```
-spec cbt_inserter_insert(Val :: integer()) -> integer().
cbt_inserter_insert(Val) ->
.

-spec cbt_inserter_get_root() -> #tree_node{} | null.
cbt_inserter_get_root() ->
.


%% Your functions will be called as such:
%% cbt_inserter_init_(Root),
%% Param_1 = cbt_inserter_insert(Val),
%% Param_2 = cbt_inserter_get_root(),

%% cbt_inserter_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 cbt-inserter%
(class object%
(super-new)

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

```
root)

; insert : exact-integer? -> exact-integer?
(define/public (insert val)
)
; get_root : -> (or/c tree-node? #f)
(define/public (get_root)
)))

;; Your cbt-inserter% object will be instantiated and called as such:
;; (define obj (new cbt-inserter% [root root]))
;; (define param_1 (send obj insert val))
;; (define param_2 (send obj get_root))
```