

# Problem 623: Add One Row to Tree

## Problem Information

**Difficulty:** Medium

**Acceptance Rate:** 0.00%

**Paid Only:** No

## Problem Description

Given the

root

of a binary tree and two integers

val

and

depth

, add a row of nodes with value

val

at the given depth

depth

.

Note that the

root

node is at depth

1

.

The adding rule is:

Given the integer

depth

, for each not null tree node

cur

at the depth

depth - 1

, create two tree nodes with value

val

as

cur

's left subtree root and right subtree root.

cur

's original left subtree should be the left subtree of the new left subtree root.

cur

's original right subtree should be the right subtree of the new right subtree root.

If

depth == 1

that means there is no depth

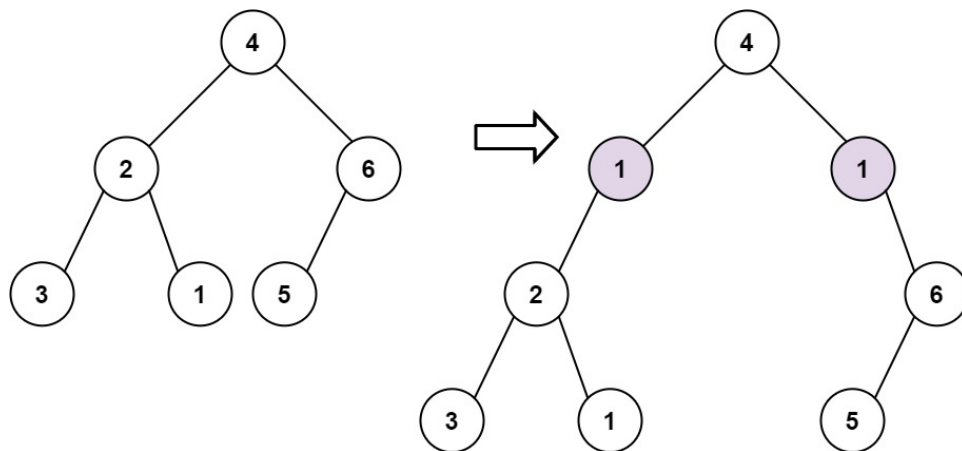
depth - 1

at all, then create a tree node with value

val

as the new root of the whole original tree, and the original tree is the new root's left subtree.

Example 1:



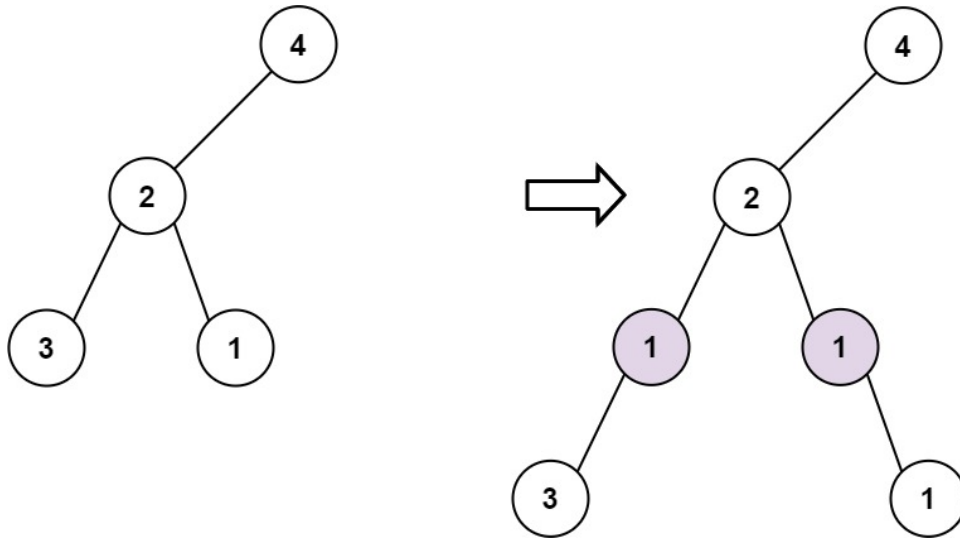
Input:

root = [4,2,6,3,1,5], val = 1, depth = 2

Output:

[4,1,1,2,null,null,6,3,1,5]

Example 2:



Input:

root = [4,2,null,3,1], val = 1, depth = 3

Output:

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

Constraints:

The number of nodes in the tree is in the range

[1, 10

4

]

.

The depth of the tree is in the range

[1, 10

4

]

.

-100 <= Node.val <= 100

-10

5

<= val <= 10

5

1 <= depth <= the depth of tree + 1

## 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 Solution {
public:
    TreeNode* addOneRow(TreeNode* root, int val, int depth) {

    }
};
```

### 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:
    TreeNode addOneRow(TreeNode root, int val, int depth) {

    }
}

```

### 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 Solution:
    def addOneRow(self, root: Optional[TreeNode], val: int, depth: int) -> Optional[TreeNode]:

```

### 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 addOneRow(self, root, val, depth):

```

```

"""
:type root: Optional[TreeNode]
:type val: int
:type depth: int
:rtype: Optional[TreeNode]
"""

```

## 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
 * @param {number} val
 * @param {number} depth
 * @return {TreeNode}
 */
var addOneRow = function(root, val, depth) {

};

```

## 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 addOneRow(root: TreeNode | null, val: number, depth: number):
TreeNode | null {

};

```

## 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 TreeNode AddOneRow(TreeNode root, int val, int depth) {

    }
}

```

## C:

```

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

}

```

## Go:

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

}
```

## 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 addOneRow(root: TreeNode?, `val`: Int, depth: Int): TreeNode? {

    }
}
```

## 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 addOneRow(_ root: TreeNode?, _ val: Int, _ depth: Int) -> TreeNode? {

}
}

```

## 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 add_one_row(root: Option<Rc<RefCell<TreeNode>>>, val: i32, depth: i32)
    -> Option<Rc<RefCell<TreeNode>>> {

    }
}

```

## 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
# @param {Integer} val
# @param {Integer} depth
# @return {TreeNode}
def add_one_row(root, val, depth)

end
```

## 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
 * @param Integer $val
 * @param Integer $depth
 * @return TreeNode
 */
}
```

```
function addOneRow($root, $val, $depth) {

}

}
```

### 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 {
  TreeNode? addOneRow(TreeNode? root, int val, int depth) {

  }
}
```

### 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 addOneRow(root: TreeNode, `val`: Int, depth: Int): TreeNode = {

  }
}
```

### 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 add_one_row(root :: TreeNode.t() | nil, val :: integer, depth :: integer)
    :: TreeNode.t() | nil
  def add_one_row(root, val, depth) do

end
end

```

## Erlang:

```

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

-spec add_one_row(Root :: #tree_node{} | null, Val :: integer(), Depth ::
integer()) -> #tree_node{} | null.
add_one_row(Root, Val, Depth) ->
.

```

## 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 (add-one-row root val depth)
  (-> (or/c tree-node? #f) exact-integer? exact-integer? (or/c tree-node? #f))
  )

```

## Solutions

### C++ Solution:

```

/*
 * Problem: Add One Row to Tree
 * 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) {}
 *   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:
  TreeNode* addOneRow(TreeNode* root, int val, int depth) {

```

```
}  
};
```

### Java Solution:

```
/**  
 * Problem: Add One Row to Tree  
 * 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 Solution {  
    public TreeNode addOneRow(TreeNode root, int val, int depth) {  
  
    }  
}
```

### Python3 Solution:

```

"""
Problem: Add One Row to Tree
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 Solution:
def addOneRow(self, root: Optional[TreeNode], val: int, depth: int) ->
Optional[TreeNode]:
# 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 Solution(object):
def addOneRow(self, root, val, depth):
"""
:type root: Optional[TreeNode]
:type val: int
:type depth: int
:rtype: Optional[TreeNode]
"""

```

### JavaScript Solution:

```

/**
 * Problem: Add One Row to Tree
 * 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
 * @param {number} val
 * @param {number} depth
 * @return {TreeNode}
 */
var addOneRow = function(root, val, depth) {

};

```

## TypeScript Solution:

```

/**
 * Problem: Add One Row to Tree
 * 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)
*   }
* }
*/

function addOneRow(root: TreeNode | null, val: number, depth: number):
TreeNode | null {

};

```

## C# Solution:

```

/*
* Problem: Add One Row to Tree
* 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 Solution {
    public TreeNode AddOneRow(TreeNode root, int val, int depth) {

    }
}

```

### C Solution:

```

/*
 * Problem: Add One Row to Tree
 * 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;
 * };
 */
struct TreeNode* addOneRow(struct TreeNode* root, int val, int depth) {

}

```

### Go Solution:

```

// Problem: Add One Row to Tree
// 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
 * }
 */
func addOneRow(root *TreeNode, val int, depth int) *TreeNode {

}

```

### 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 Solution {
    fun addOneRow(root: TreeNode?, `val`: Int, depth: Int): TreeNode? {

    }
}

```

### 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 addOneRow(_ root: TreeNode?, _ val: Int, _ depth: Int) -> TreeNode? {

}
}

```

## Rust Solution:

```

// Problem: Add One Row to Tree
// 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
//         }
//     }
// }

```

```

use std::rc::Rc;
use std::cell::RefCell;
impl Solution {
pub fn add_one_row(root: Option<Rc<RefCell<TreeNode>>>, val: i32, depth: i32)
-> Option<Rc<RefCell<TreeNode>>> {

}
}

```

### 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
# @param {TreeNode} root
# @param {Integer} val
# @param {Integer} depth
# @return {TreeNode}
def add_one_row(root, val, depth)

end

```

### 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 Solution {

/**
 * @param TreeNode $root
 * @param Integer $val
 * @param Integer $depth
 * @return TreeNode
 */
function addOneRow($root, $val, $depth) {

}

}

```

### 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 Solution {
  TreeNode? addOneRow(TreeNode? root, int val, int depth) {

  }

}

```

### 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 addOneRow(root: TreeNode, `val`: Int, depth: Int): TreeNode = {

  }
}

```

### 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 add_one_row(root :: TreeNode.t() | nil, val :: integer, depth :: integer)
    :: TreeNode.t() | nil
  def add_one_row(root, val, depth) do

  end
end

```

### 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 add_one_row(Root :: #tree_node{} | null, Val :: integer(), Depth ::
integer()) -> #tree_node{} | null.
add_one_row(Root, Val, Depth) ->
.

```

## 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/contract (add-one-row root val depth)
  (-> (or/c tree-node? #f) exact-integer? exact-integer? (or/c tree-node? #f))
  )
```