

Problem 105: Construct Binary Tree from Preorder and Inorder Traversal

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Given two integer arrays

preorder

and

inorder

where

preorder

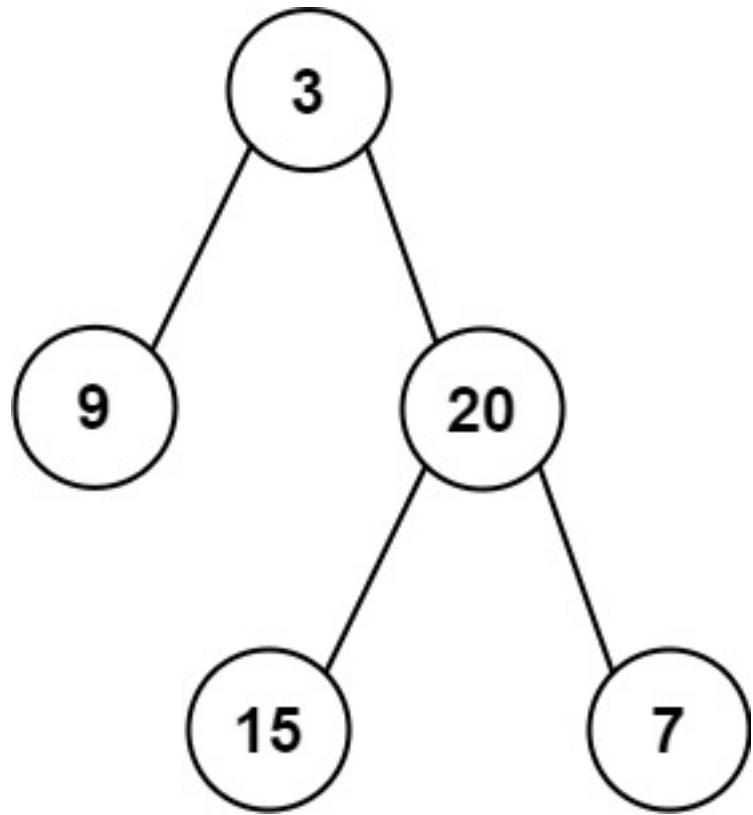
is the preorder traversal of a binary tree and

inorder

is the inorder traversal of the same tree, construct and return

the binary tree

Example 1:



Input:

preorder = [3,9,20,15,7], inorder = [9,3,15,20,7]

Output:

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

Example 2:

Input:

preorder = [-1], inorder = [-1]

Output:

[-1]

Constraints:

1 <= preorder.length <= 3000

inorder.length == preorder.length

-3000 <= preorder[i], inorder[i] <= 3000

preorder

and

inorder

consist of

unique

values.

Each value of

inorder

also appears in

preorder

.

preorder

is

guaranteed

to be the preorder traversal of the tree.

inorder

is

guaranteed

to be the inorder traversal 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 Solution {
public:
    TreeNode* buildTree(vector<int>& preorder, vector<int>& inorder) {
        }
    };
}
```

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 buildTree(int[] preorder, int[] inorder) {

}
}

```

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 buildTree(self, preorder: List[int], inorder: List[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 buildTree(self, preorder, inorder):
        """
:type preorder: List[int]
:type inorder: List[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 {number[]} preorder
* @param {number[]} inorder
* @return {TreeNode}
*/
var buildTree = function(preorder, inorder) {

};


```

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 buildTree(preorder: number[], inorder: 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 BuildTree(int[] preorder, int[] inorder) {

}
}

```

C:

```

/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *   int val;
 *   struct TreeNode *left;
 *   struct TreeNode *right;
 * };
 */
struct TreeNode* buildTree(int* preorder, int preorderSize, int* inorder, int
inorderSize) {

}

```

Go:

```

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *   Val int
 *   Left *TreeNode
 *   Right *TreeNode
 * }
 */
func buildTree(preorder []int, inorder []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 buildTree(preorder: IntArray, inorder: IntArray): 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 buildTree(_ preorder: [Int], _ inorder: [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 build_tree(preorder: Vec<i32>, inorder: Vec<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 {Integer[]} preorder
# @param {Integer[]} inorder
# @return {TreeNode}
```

```
def build_tree(preorder, inorder)

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 Integer[] $preorder
 * @param Integer[] $inorder
 * @return TreeNode
 */
function buildTree($preorder, $inorder) {

}

}
```

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? buildTree(List<int> preorder, List<int> inorder) {
        ...
    }
}

```

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 buildTree(preorder: Array[Int], inorder: Array[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 build_tree([integer], [integer]) :: TreeNode.t | nil
  def build_tree(preorder, inorder) 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 build_tree(Preorder :: [integer()], Inorder :: [integer()]) ->  
#tree_node{} | null.  
build_tree(Preorder, Inorder) ->  
.  
.
```

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 (build-tree preorder inorder)  
(-> (listof exact-integer?) (listof exact-integer?) (or/c tree-node? #f)))  
)
```

Solutions

C++ Solution:

```

/*
 * Problem: Construct Binary Tree from Preorder and Inorder Traversal
 * Difficulty: Medium
 * Tags: array, tree, hash
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * 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* buildTree(vector<int>& preorder, vector<int>& inorder) {
}
};

```

Java Solution:

```

/**
 * Problem: Construct Binary Tree from Preorder and Inorder Traversal
 * Difficulty: Medium
 * Tags: array, tree, hash
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * 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 buildTree(int[] preorder, int[] inorder) {

}
}

```

Python3 Solution:

```

"""
Problem: Construct Binary Tree from Preorder and Inorder Traversal
Difficulty: Medium
Tags: array, tree, hash

```

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
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 buildTree(self, preorder: List[int], inorder: List[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 buildTree(self, preorder, inorder):
        """
        :type preorder: List[int]
        :type inorder: List[int]
        :rtype: Optional[TreeNode]
        """

```

JavaScript Solution:

```

/**
 * Problem: Construct Binary Tree from Preorder and Inorder Traversal
 * Difficulty: Medium
 * Tags: array, tree, hash
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * 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 {number[]} preorder
* @param {number[]} inorder
* @return {TreeNode}
*/
var buildTree = function(preorder, inorder) {

};

```

TypeScript Solution:

```

/**
* Problem: Construct Binary Tree from Preorder and Inorder Traversal
* Difficulty: Medium
* Tags: array, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* 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 buildTree(preorder: number[], inorder: number[]): TreeNode | null {

};

```

C# Solution:

```
/*
 * Problem: Construct Binary Tree from Preorder and Inorder Traversal
 * Difficulty: Medium
 * Tags: array, tree, hash
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * 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 BuildTree(int[] preorder, int[] inorder) {
        }

    }
}
```

C Solution:

```
/*
 * Problem: Construct Binary Tree from Preorder and Inorder Traversal
 * Difficulty: Medium
 * Tags: array, tree, hash
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * 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* buildTree(int* preorder, int preorderSize, int* inorder, int
inorderSize) {

}

```

Go Solution:

```

// Problem: Construct Binary Tree from Preorder and Inorder Traversal
// Difficulty: Medium
// Tags: array, tree, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// 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 buildTree(preorder []int, inorder []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 buildTree(preorder: IntArray, inorder: IntArray): 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 buildTree(_ preorder: [Int], _ inorder: [Int]) -> TreeNode? {
}
}

```

Rust Solution:

```

// Problem: Construct Binary Tree from Preorder and Inorder Traversal
// Difficulty: Medium
// Tags: array, tree, hash

```

```

// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// 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 build_tree(preorder: Vec<i32>, inorder: Vec<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 {Integer[]} preorder
# @param {Integer[]} inorder
# @return {TreeNode}
def build_tree(preorder, inorder)

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 Integer[] $preorder
 * @param Integer[] $inorder
 * @return TreeNode
 */
function buildTree($preorder, $inorder) {

}
}

```

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? buildTree(List<int> preorder, List<int> inorder) {
}

}

```

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 buildTree(preorder: Array[Int], inorder: Array[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 build_tree(preorder :: [integer], inorder :: [integer]) :: TreeNode.t | nil
def build_tree(preorder, inorder) 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 build_tree(Preorder :: [integer()], Inorder :: [integer()]) -> #tree_node{} | null.
build_tree(Preorder, Inorder) ->
.
.
```

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 (build-tree preorder inorder)
  (-> (listof exact-integer?) (listof exact-integer?) (or/c tree-node? #f)))

```

