Problem 889: Construct Binary Tree from Preorder and Postorder Traversal

Problem Information

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

Problem Description

Given two integer arrays,

preorder

and

postorder

where

preorder

is the preorder traversal of a binary tree of

distinct

values and

postorder

is the postorder traversal of the same tree, reconstruct and return

the binary tree

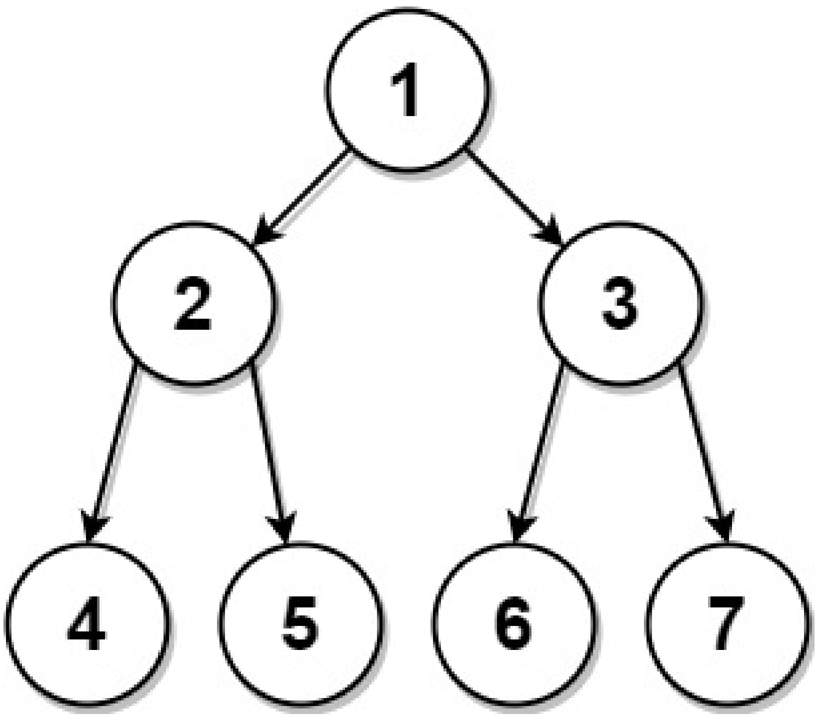
.

If there exist multiple answers, you can

return any

of them.

Example 1:



Input:

preorder = [1,2,4,5,3,6,7], postorder = [4,5,2,6,7,3,1]

Output:

[1,2,3,4,5,6,7]

Example 2:

Input:

preorder = [1], postorder = [1]

Output:

[1]

Constraints:

1 <= preorder.length <= 30

1 <= preorder[i] <= preorder.length

All the values of

preorder

are

unique

.

postorder.length == preorder.length

1 <= postorder[i] <= postorder.length

All the values of

postorder

are

unique

.

It is guaranteed that

preorder

and

postorder

are the preorder traversal and postorder traversal of the same binary 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* constructFromPrePost(vector<int>& preorder, vector<int>& postorder)
{

}
};
```

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 constructFromPrePost(int[] preorder, int[] postorder) {

}
}
```

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 constructFromPrePost(self, preorder: List[int], postorder: 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 constructFromPrePost(self, preorder, postorder):
"""
:type preorder: List[int]
:type postorder: 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[]} postorder
* @return {TreeNode}
*/
var constructFromPrePost = function(preorder, postorder) {

};
```

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 constructFromPrePost(preorder: number[], postorder: 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 ConstructFromPrePost(int[] preorder, int[] postorder) {

}
}
```

C:

```
/**
* Definition for a binary tree node.
* struct TreeNode {
* int val;
* struct TreeNode *left;
* struct TreeNode *right;
* };
*/
struct TreeNode* constructFromPrePost(int* preorder, int preorderSize, int*
postorder, int postorderSize) {

}
```

Go:

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

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[] $postorder
* @return TreeNode
*/
function constructFromPrePost($preorder, $postorder) {

}
}
```

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? constructFromPrePost(List<int> preorder, List<int> postorder) {

}
}
```

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 constructFromPrePost(preorder: Array[Int], postorder: 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 construct_from_pre_post(preorder :: [integer], postorder :: [integer])
:: TreeNode.t | nil
def construct_from_pre_post(preorder, postorder) 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 construct_from_pre_post(Preorder :: [integer()], Postorder ::
[integer()]) -> #tree_node{} | null.
construct_from_pre_post(Preorder, Postorder) ->
.
```

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 (construct-from-pre-post preorder postorder)
(-> (listof exact-integer?) (listof exact-integer?) (or/c tree-node? #f))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Construct Binary Tree from Preorder and Postorder 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* constructFromPrePost(vector<int>& preorder, vector<int>& postorder)
{

}
};
```

Java Solution:

```
/**
* Problem: Construct Binary Tree from Preorder and Postorder 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 constructFromPrePost(int[] preorder, int[] postorder) {

}
}
```

Python3 Solution:

```
"""
Problem: Construct Binary Tree from Preorder and Postorder 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 constructFromPrePost(self, preorder: List[int], postorder: 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 constructFromPrePost(self, preorder, postorder):
"""
:type preorder: List[int]
:type postorder: List[int]
:rtype: Optional[TreeNode]
"""
```

JavaScript Solution:

```
/**
* Problem: Construct Binary Tree from Preorder and Postorder 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[]} postorder
* @return {TreeNode}
*/
var constructFromPrePost = function(preorder, postorder) {

};
```

TypeScript Solution:

```
/**
* Problem: Construct Binary Tree from Preorder and Postorder 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 constructFromPrePost(preorder: number[], postorder: number[]):
TreeNode | null {

};
```

C# Solution:

```
/*
* Problem: Construct Binary Tree from Preorder and Postorder 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 ConstructFromPrePost(int[] preorder, int[] postorder) {

}
}
```

C Solution:

```
/*
* Problem: Construct Binary Tree from Preorder and Postorder 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* constructFromPrePost(int* preorder, int preorderSize, int*
postorder, int postorderSize) {

}
```

Go Solution:

```
// Problem: Construct Binary Tree from Preorder and Postorder 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 constructFromPrePost(preorder []int, postorder []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 constructFromPrePost(preorder: IntArray, postorder: 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 constructFromPrePost(_ preorder: [Int], _ postorder: [Int]) -> TreeNode?
{

}
}
```

Rust Solution:

```
// Problem: Construct Binary Tree from Preorder and Postorder 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 construct_from_pre_post(preorder: Vec<i32>, postorder: 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[]} postorder
# @return {TreeNode}
def construct_from_pre_post(preorder, postorder)

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[] $postorder
* @return TreeNode
```

```
*/
function constructFromPrePost($preorder, $postorder) {

}
}
```

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? constructFromPrePost(List<int> preorder, List<int> postorder) {

}
}
```

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 constructFromPrePost(preorder: Array[Int], postorder: 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 construct_from_pre_post(preorder :: [integer], postorder :: [integer])
:: TreeNode.t | nil
def construct_from_pre_post(preorder, postorder) 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 construct_from_pre_post(Preorder :: [integer()], Postorder ::
[integer()]) -> #tree_node{} | null.
construct_from_pre_post(Preorder, Postorder) ->
.
```

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 (construct-from-pre-post preorder postorder)
(-> (listof exact-integer?) (listof exact-integer?) (or/c tree-node? #f))
)
```