# Problem 1430: Check If a String Is a Valid Sequence from Root to Leaves Path in a Binary Tree

## Problem Information

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

## Problem Description

Given a binary tree where each path going from the root to any leaf form a

valid sequence

, check if a given string is a

valid sequence

in such binary tree.

We get the given string from the concatenation of an array of integers

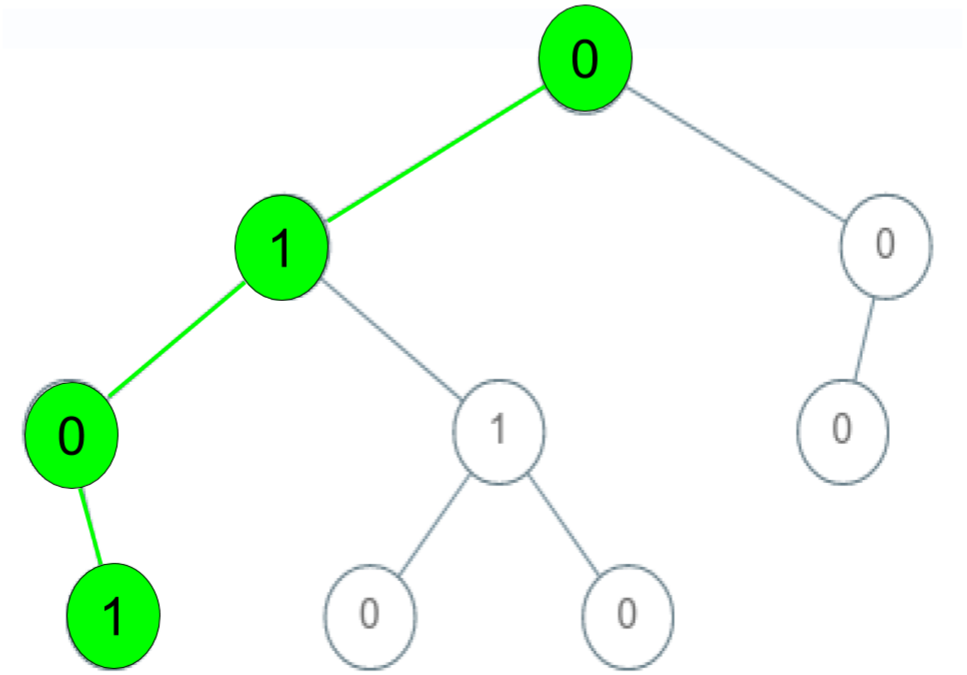
arr

and the concatenation of all values of the nodes along a path results in a

sequence

in the given binary tree.

Example 1:



Input:

root = [0,1,0,0,1,0,null,null,1,0,0], arr = [0,1,0,1]

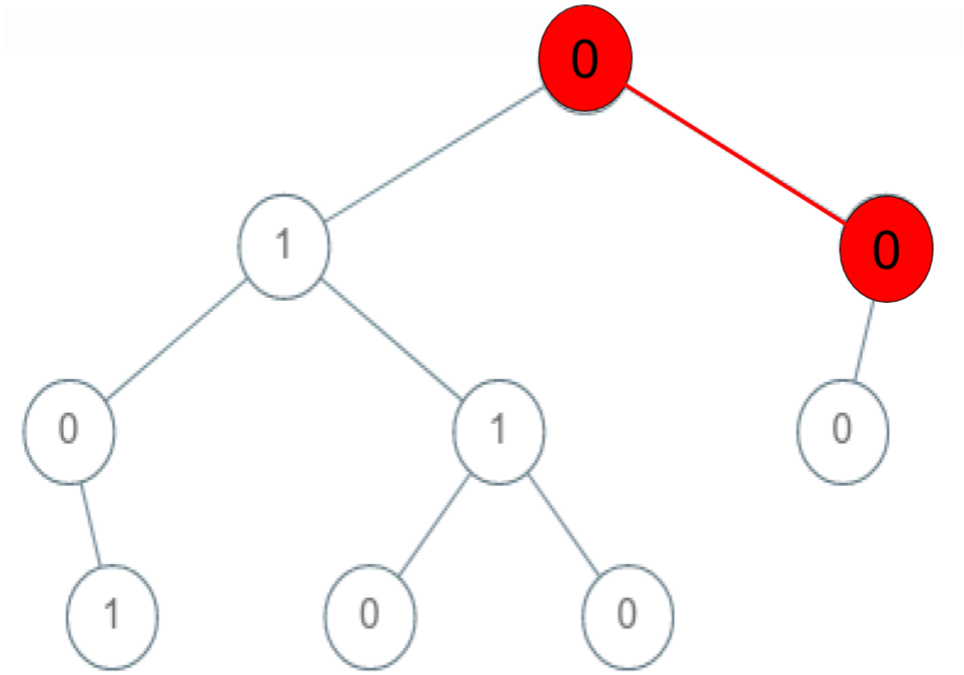
Output:

true

Explanation:

The path 0 -> 1 -> 0 -> 1 is a valid sequence (green color in the figure). Other valid sequences are: 0 -> 1 -> 1 -> 0 0 -> 0 -> 0

Example 2:



Input:

root = [0,1,0,0,1,0,null,null,1,0,0], arr = [0,0,1]

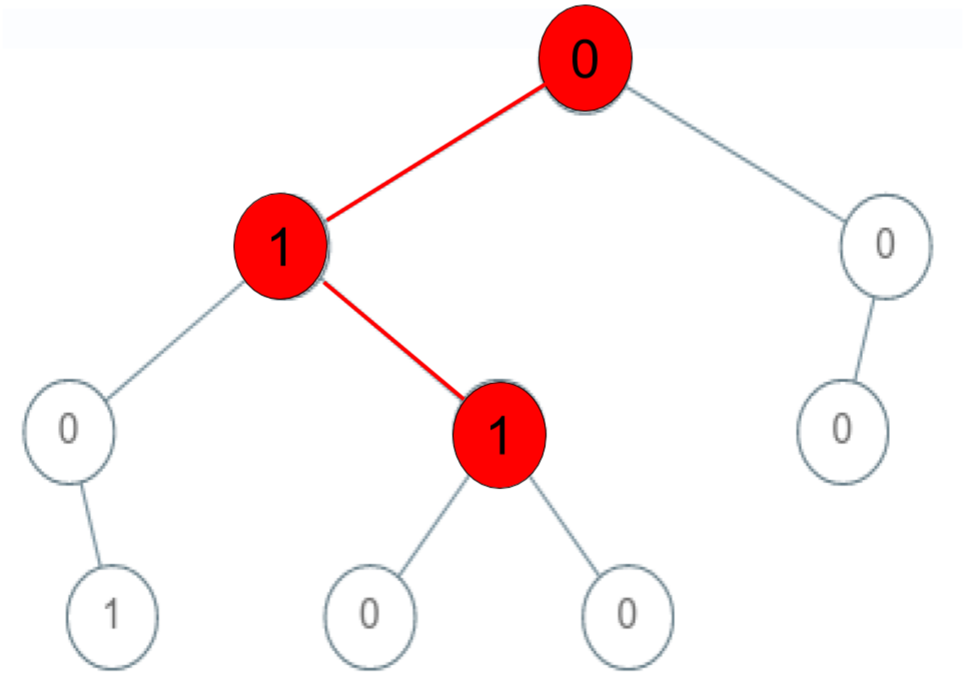
Output:

false

Explanation:

The path 0 -> 0 -> 1 does not exist, therefore it is not even a sequence.

Example 3:



Input:

root = [0,1,0,0,1,0,null,null,1,0,0], arr = [0,1,1]

Output:

false

Explanation:

The path 0 -> 1 -> 1 is a sequence, but it is not a valid sequence.

Constraints:

1 <= arr.length <= 5000

0 <= arr[i] <= 9

Each node's value is between [0 - 9].

## 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:
bool isValidSequence(TreeNode* root, vector<int>& arr) {

}
};
```

**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 boolean isValidSequence(TreeNode root, int[] arr) {

}
}
```

### 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 isValidSequence(self, root: Optional[TreeNode], arr: List[int]) -> bool:
```

### 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 isValidSequence(self, root, arr):
"""
:type root: Optional[TreeNode]
:type arr: List[int]
:rtype: bool
"""
```

### 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[]} arr
* @return {boolean}
*/
var isValidSequence = function(root, arr) {



};
```

### 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 isValidSequence(root: TreeNode | null, arr: number[]): boolean {

};
```

### 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 bool IsValidSequence(TreeNode root, int[] arr) {

}
}
```

**C:**

```
/**
* Definition for a binary tree node.
* struct TreeNode {
* int val;
* struct TreeNode *left;
* struct TreeNode *right;
* };
*/
bool isValidSequence(struct TreeNode* root, int* arr, int arrSize) {

}
```

**Go:**

```
/**
* Definition for a binary tree node.
* type TreeNode struct {
* Val int
* Left *TreeNode
* Right *TreeNode
* }
*/
func isValidSequence(root *TreeNode, arr []int) bool {

}
```

**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 isValidSequence(root: TreeNode?, arr: IntArray): Boolean {

}
}
```

**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 isValidSequence(_ root: TreeNode?, _ arr: [Int]) -> Bool {

}
}
```

**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 is_valid_sequence(root: Option<Rc<RefCell<TreeNode>>>, arr: Vec<i32>)
-> bool {

}
}
```

### 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[]} arr
# @return {Boolean}
def is_valid_sequence(root, arr)

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[] $arr
* @return Boolean
*/
function isValidSequence($root, $arr) {

}
}
```

**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 {
bool isValidSequence(TreeNode? root, List<int> arr) {

}
}
```

**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 isValidSequence(root: TreeNode, arr: Array[Int]): Boolean = {

}
}
```

### 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 is_valid_sequence(root :: TreeNode.t | nil, arr :: [integer]) ::
boolean
def is_valid_sequence(root, arr) 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 is_valid_sequence(Root :: #tree_node{} | null, Arr :: [integer()]) ->
boolean().
is_valid_sequence(Root, Arr) ->
.
```

### 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 (is-valid-sequence root arr)
(-> (or/c tree-node? #f) (listof exact-integer?) boolean?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Check If a String Is a Valid Sequence from Root to Leaves Path in
a Binary Tree
* Difficulty: Medium
* Tags: array, string, tree, search
*
* 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) {
// 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 Solution {
public:
bool isValidSequence(TreeNode* root, vector<int>& arr) {

}
};
```

### Java Solution:

```
/**
* Problem: Check If a String Is a Valid Sequence from Root to Leaves Path in
a Binary Tree
* Difficulty: Medium
* Tags: array, string, tree, search
*
* 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 boolean isValidSequence(TreeNode root, int[] arr) {

}
}
```

### Python3 Solution:

```
"""
Problem: Check If a String Is a Valid Sequence from Root to Leaves Path in a Binary Tree
Difficulty: Medium
Tags: array, string, tree, search

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 isValidSequence(self, root: Optional[TreeNode], arr: List[int]) -> bool:
# 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 isValidSequence(self, root, arr):
"""
:type root: Optional[TreeNode]
:type arr: List[int]
:rtype: bool
"""
```

### JavaScript Solution:

```
/**
* Problem: Check If a String Is a Valid Sequence from Root to Leaves Path in
a Binary Tree
* Difficulty: Medium
* Tags: array, string, tree, search
*
* 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 {TreeNode} root
* @param {number[]} arr
* @return {boolean}
*/
var isValidSequence = function(root, arr) {

};
```

### TypeScript Solution:

```
/**
* Problem: Check If a String Is a Valid Sequence from Root to Leaves Path in
a Binary Tree
* Difficulty: Medium
* Tags: array, string, tree, search
*
* 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 isValidSequence(root: TreeNode | null, arr: number[]): boolean {

};
```

### C# Solution:

```
/*
* Problem: Check If a String Is a Valid Sequence from Root to Leaves Path in
a Binary Tree
* Difficulty: Medium
* Tags: array, string, tree, search
*
* 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 bool IsValidSequence(TreeNode root, int[] arr) {

}
}
```

### C Solution:

```
/*
* Problem: Check If a String Is a Valid Sequence from Root to Leaves Path in
a Binary Tree
```

```
* Difficulty: Medium
* Tags: array, string, tree, search
*
* 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;
* };
*/
bool isValidSequence(struct TreeNode* root, int* arr, int arrSize) {

}
```

### Go Solution:

```
// Problem: Check If a String Is a Valid Sequence from Root to Leaves Path in
a Binary Tree
// Difficulty: Medium
// Tags: array, string, tree, search
//
// 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 isValidSequence(root *TreeNode, arr []int) bool {
}
```

### 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 isValidSequence(root: TreeNode?, arr: IntArray): Boolean {

}
}
```

### 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 isValidSequence(_ root: TreeNode?, _ arr: [Int]) -> Bool {
}
}
```

### Rust Solution:

```
// Problem: Check If a String Is a Valid Sequence from Root to Leaves Path in a Binary Tree
// Difficulty: Medium
// Tags: array, string, tree, search
//
// 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 is_valid_sequence(root: Option<Rc<RefCell<TreeNode>>>, arr: Vec<i32>) -> bool {

}
}
```

### 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[]} arr
# @return {Boolean}
def is_valid_sequence(root, arr)

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[] $arr
* @return Boolean
*/
function isValidSequence($root, $arr) {

}
}
```

### 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 {
bool isValidSequence(TreeNode? root, List<int> arr) {

}
}
```

### 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 isValidSequence(root: TreeNode, arr: Array[Int]): Boolean = {

}
}
```

### 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 is_valid_sequence(root :: TreeNode.t | nil, arr :: [integer]) ::
boolean
def is_valid_sequence(root, arr) 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 is_valid_sequence(Root :: #tree_node{} | null, Arr :: [integer()]) ->
boolean().
is_valid_sequence(Root, Arr) ->
.
```

### 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 (is-valid-sequence root arr)
(-> (or/c tree-node? #f) (listof exact-integer?) boolean?)
)
```