Problem 331: Verify Preorder Serialization of a Binary Tree

Problem Information

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

Problem Description

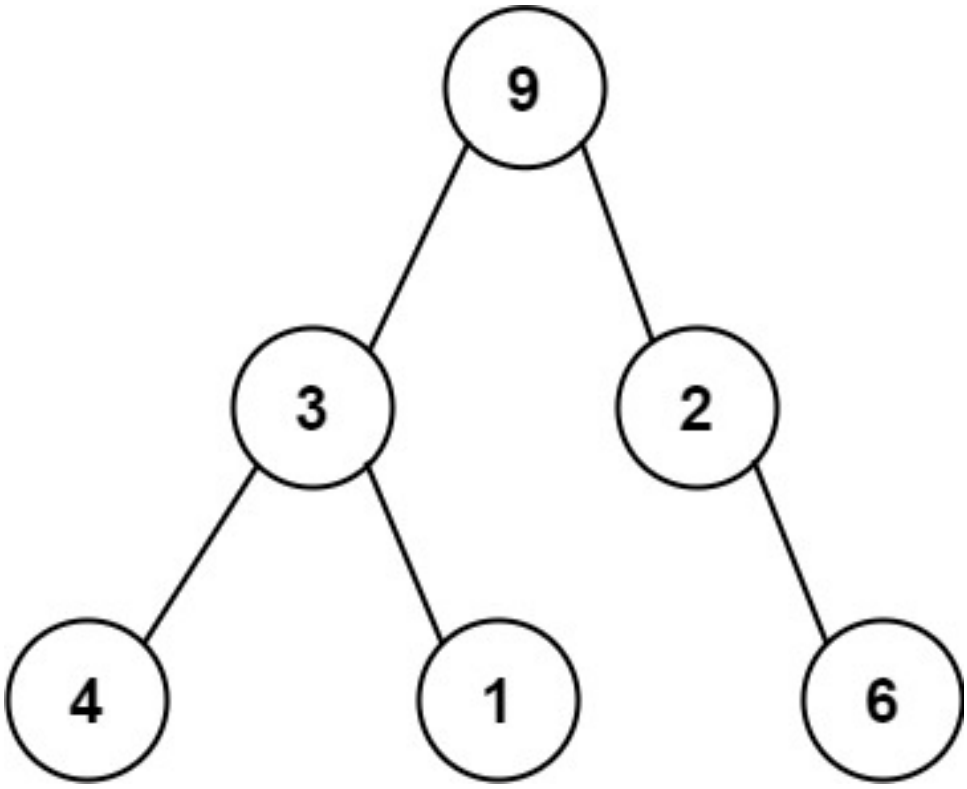
One way to serialize a binary tree is to use

preorder traversal

. When we encounter a non-null node, we record the node's value. If it is a null node, we record using a sentinel value such as

'#'

.



For example, the above binary tree can be serialized to the string

"9,3,4,#,#,1,#,#,2,#,6,#,#"

, where

'#'

represents a null node.

Given a string of comma-separated values

preorder

, return

true

if it is a correct preorder traversal serialization of a binary tree.

It is

guaranteed

that each comma-separated value in the string must be either an integer or a character

'#'

representing null pointer.

You may assume that the input format is always valid.

For example, it could never contain two consecutive commas, such as

"1,,3"

.

Note:

You are not allowed to reconstruct the tree.

Example 1:

Input:

preorder = "9,3,4,#,#,1,#,#,2,#,6,#,#"

Output:

true

Example 2:

Input:

preorder = "1,#"

Output:

false

Example 3:

Input:

preorder = "9,#,#,1"

Output:

false

Constraints:

1 <= preorder.length <= 10

4

preorder

consist of integers in the range

[0, 100]

and

'#'

separated by commas

','

.

Code Snippets

C++:

```
class Solution {
public:
bool isValidSerialization(string preorder) {

}
};
```

Java:

```
class Solution {
public boolean isValidSerialization(String preorder) {

}
}
```

Python3:

```
class Solution:
def isValidSerialization(self, preorder: str) -> bool:
```

Python:

```
class Solution(object):
def isValidSerialization(self, preorder):
"""
:type preorder: str
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {string} preorder
* @return {boolean}
*/
var isValidSerialization = function(preorder) {

};
```

TypeScript:

```
function isValidSerialization(preorder: string): boolean {
```

```
};
```

C#:

```
public class Solution {
public bool IsValidSerialization(string preorder) {

}
}
```

C:

```
bool isValidSerialization(char* preorder) {

}
```

Go:

```
func isValidSerialization(preorder string) bool {

}
```

Kotlin:

```
class Solution {
fun isValidSerialization(preorder: String): Boolean {

}
}
```

Swift:

```
class Solution {
func isValidSerialization(_ preorder: String) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn is_valid_serialization(preorder: String) -> bool {
```

```
}
}
```

Ruby:

```
# @param {String} preorder
# @return {Boolean}
def is_valid_serialization(preorder)

end
```

PHP:

```
class Solution {

/**
* @param String $preorder
* @return Boolean
*/
function isValidSerialization($preorder) {

}
}
```

Dart:

```
class Solution {
bool isValidSerialization(String preorder) {

}
}
```

Scala:

```
object Solution {
def isValidSerialization(preorder: String): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec is_valid_serialization(preorder :: String.t) :: boolean
def is_valid_serialization(preorder) do

end
end
```

Erlang:

```
-spec is_valid_serialization(Preorder :: unicode:unicode_binary()) ->
boolean().
is_valid_serialization(Preorder) ->
.
```

Racket:

```
(define/contract (is-valid-serialization preorder)
(-> string? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Verify Preorder Serialization of a Binary Tree
* Difficulty: Medium
* Tags: string, tree, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
bool isValidSerialization(string preorder) {

}
};
```

Java Solution:

```
/**
* Problem: Verify Preorder Serialization of a Binary Tree
* Difficulty: Medium
* Tags: string, tree, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public boolean isValidSerialization(String preorder) {

}
}
```

Python3 Solution:

```
"""
Problem: Verify Preorder Serialization of a Binary Tree
Difficulty: Medium
Tags: string, tree, stack

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
def isValidSerialization(self, preorder: str) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def isValidSerialization(self, preorder):
"""
:type preorder: str
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Verify Preorder Serialization of a Binary Tree
* Difficulty: Medium
* Tags: string, tree, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {string} preorder
* @return {boolean}
*/
var isValidSerialization = function(preorder) {

};
```

TypeScript Solution:

```
/**
* Problem: Verify Preorder Serialization of a Binary Tree
* Difficulty: Medium
* Tags: string, tree, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function isValidSerialization(preorder: string): boolean {

};
```

C# Solution:

```
/*
* Problem: Verify Preorder Serialization of a Binary Tree
* Difficulty: Medium
* Tags: string, tree, stack
```

```
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public bool IsValidSerialization(string preorder) {

}
}
```

C Solution:

```
/*
* Problem: Verify Preorder Serialization of a Binary Tree
* Difficulty: Medium
* Tags: string, tree, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

bool isValidSerialization(char* preorder) {

}
```

Go Solution:

```
// Problem: Verify Preorder Serialization of a Binary Tree
// Difficulty: Medium
// Tags: string, tree, stack
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func isValidSerialization(preorder string) bool {

}
```

Kotlin Solution:

```
class Solution {
fun isValidSerialization(preorder: String): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func isValidSerialization(_ preorder: String) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Verify Preorder Serialization of a Binary Tree
// Difficulty: Medium
// Tags: string, tree, stack
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn is_valid_serialization(preorder: String) -> bool {

}
}
```

Ruby Solution:

```
# @param {String} preorder
# @return {Boolean}
def is_valid_serialization(preorder)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $preorder
* @return Boolean
*/
function isValidSerialization($preorder) {

}
}
```

Dart Solution:

```
class Solution {
bool isValidSerialization(String preorder) {

}
}
```

Scala Solution:

```
object Solution {
def isValidSerialization(preorder: String): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec is_valid_serialization(preorder :: String.t) :: boolean
def is_valid_serialization(preorder) do

end
end
```

Erlang Solution:

```
-spec is_valid_serialization(Preorder :: unicode:unicode_binary()) ->
boolean().
is_valid_serialization(Preorder) ->
.
```

Racket Solution:

```
(define/contract (is-valid-serialization preorder)
(-> string? boolean?)
)
```