Problem 1361: Validate Binary Tree Nodes

Problem Information

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

Problem Description

You have

n

binary tree nodes numbered from

0

to

n - 1

where node

i

has two children

leftChild[i]

and

rightChild[i]

, return

true

if and only if

all

the given nodes form

exactly one

valid binary tree.

If node

i

has no left child then

leftChild[i]

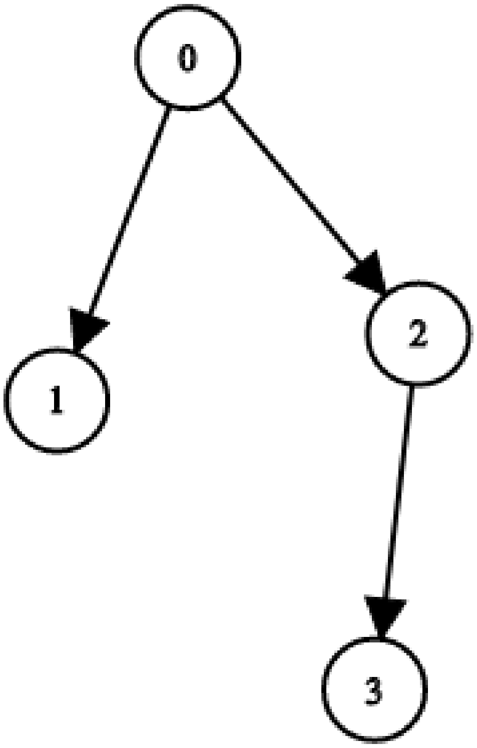
will equal

-1

, similarly for the right child.

Note that the nodes have no values and that we only use the node numbers in this problem.

Example 1:



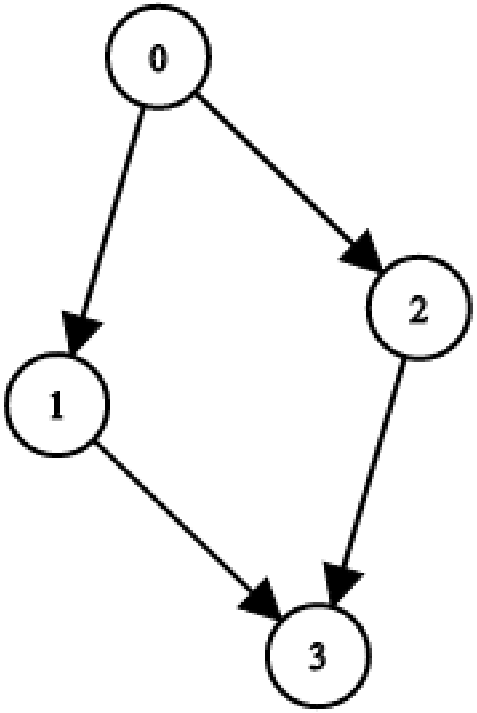
Input:

n = 4, leftChild = [1,-1,3,-1], rightChild = [2,-1,-1,-1]

Output:

true

Example 2:



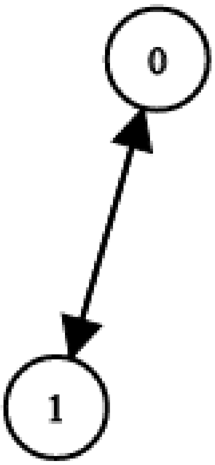
Input:

n = 4, leftChild = [1,-1,3,-1], rightChild = [2,3,-1,-1]

Output:

false

Example 3:



Input:

n = 2, leftChild = [1,0], rightChild = [-1,-1]

Output:

false

Constraints:

n == leftChild.length == rightChild.length

1 <= n <= 10

4

-1 <= leftChild[i], rightChild[i] <= n - 1

Code Snippets

C++:

```
class Solution {
public:
bool validateBinaryTreeNodes(int n, vector<int>& leftChild, vector<int>&
rightChild) {

}
};
```

Java:

```
class Solution {
public boolean validateBinaryTreeNodes(int n, int[] leftChild, int[]
rightChild) {

}
}
```

Python3:

```
class Solution:
def validateBinaryTreeNodes(self, n: int, leftChild: List[int], rightChild:
List[int]) -> bool:
```

Python:

```
class Solution(object):
def validateBinaryTreeNodes(self, n, leftChild, rightChild):
"""
:type n: int
:type leftChild: List[int]
:type rightChild: List[int]
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[]} leftChild
* @param {number[]} rightChild
* @return {boolean}
*/
var validateBinaryTreeNodes = function(n, leftChild, rightChild) {

};
```

TypeScript:

```
function validateBinaryTreeNodes(n: number, leftChild: number[], rightChild:
number[]): boolean {

};
```

C#:

```
public class Solution {
public bool ValidateBinaryTreeNodes(int n, int[] leftChild, int[] rightChild)
{

}
}
```

C:

```
bool validateBinaryTreeNodes(int n, int* leftChild, int leftChildSize, int*
rightChild, int rightChildSize) {

}
```

Go:

```
func validateBinaryTreeNodes(n int, leftChild []int, rightChild []int) bool {

}
```

Kotlin:

```
class Solution {
fun validateBinaryTreeNodes(n: Int, leftChild: IntArray, rightChild:
IntArray): Boolean {

}
}
```

Swift:

```
class Solution {
func validateBinaryTreeNodes(_ n: Int, _ leftChild: [Int], _ rightChild:
[Int]) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn validate_binary_tree_nodes(n: i32, left_child: Vec<i32>, right_child:
Vec<i32>) -> bool {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[]} left_child
# @param {Integer[]} right_child
# @return {Boolean}
def validate_binary_tree_nodes(n, left_child, right_child)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[] $leftChild
* @param Integer[] $rightChild
* @return Boolean
*/
function validateBinaryTreeNodes($n, $leftChild, $rightChild) {

}
}
```

Dart:

```
class Solution {
bool validateBinaryTreeNodes(int n, List<int> leftChild, List<int>
rightChild) {

}
}
```

Scala:

```
object Solution {
def validateBinaryTreeNodes(n: Int, leftChild: Array[Int], rightChild:
Array[Int]): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec validate_binary_tree_nodes(n :: integer, left_child :: [integer],
right_child :: [integer]) :: boolean
def validate_binary_tree_nodes(n, left_child, right_child) do

end
end
```

Erlang:

```
-spec validate_binary_tree_nodes(N :: integer(), LeftChild :: [integer()],
RightChild :: [integer()]) -> boolean().
validate_binary_tree_nodes(N, LeftChild, RightChild) ->
.
```

Racket:

```
(define/contract (validate-binary-tree-nodes n leftChild rightChild)
(-> exact-integer? (listof exact-integer?) (listof exact-integer?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Validate Binary Tree Nodes
* Difficulty: Medium
* Tags: tree, graph, 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
*/

class Solution {
public:
bool validateBinaryTreeNodes(int n, vector<int>& leftChild, vector<int>&
rightChild) {

}
};
```

Java Solution:

```
/**
* Problem: Validate Binary Tree Nodes
* Difficulty: Medium
* Tags: tree, graph, 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
*/

class Solution {
public boolean validateBinaryTreeNodes(int n, int[] leftChild, int[]
rightChild) {

}
}
```

Python3 Solution:

```
"""
Problem: Validate Binary Tree Nodes
Difficulty: Medium
Tags: tree, graph, 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
"""

class Solution:
def validateBinaryTreeNodes(self, n: int, leftChild: List[int], rightChild:
List[int]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def validateBinaryTreeNodes(self, n, leftChild, rightChild):
"""
:type n: int
:type leftChild: List[int]
:type rightChild: List[int]
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Validate Binary Tree Nodes
* Difficulty: Medium
* Tags: tree, graph, 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
*/

/**
* @param {number} n
* @param {number[]} leftChild
* @param {number[]} rightChild
* @return {boolean}
*/
var validateBinaryTreeNodes = function(n, leftChild, rightChild) {

};
```

TypeScript Solution:

```
/**
* Problem: Validate Binary Tree Nodes
* Difficulty: Medium
* Tags: tree, graph, 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
*/
```

```
function validateBinaryTreeNodes(n: number, leftChild: number[], rightChild:
number[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Validate Binary Tree Nodes
* Difficulty: Medium
* Tags: tree, graph, 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
*/

public class Solution {
public bool ValidateBinaryTreeNodes(int n, int[] leftChild, int[] rightChild)
{

}
}
```

C Solution:

```
/*
* Problem: Validate Binary Tree Nodes
* Difficulty: Medium
* Tags: tree, graph, 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
*/

bool validateBinaryTreeNodes(int n, int* leftChild, int leftChildSize, int*
rightChild, int rightChildSize) {

}
```

Go Solution:

```
// Problem: Validate Binary Tree Nodes
// Difficulty: Medium
// Tags: tree, graph, 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

func validateBinaryTreeNodes(n int, leftChild []int, rightChild []int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun validateBinaryTreeNodes(n: Int, leftChild: IntArray, rightChild:
IntArray): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func validateBinaryTreeNodes(_ n: Int, _ leftChild: [Int], _ rightChild:
[Int]) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Validate Binary Tree Nodes
// Difficulty: Medium
// Tags: tree, graph, 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
```

```
impl Solution {
pub fn validate_binary_tree_nodes(n: i32, left_child: Vec<i32>, right_child:
Vec<i32>) -> bool {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[]} left_child
# @param {Integer[]} right_child
# @return {Boolean}
def validate_binary_tree_nodes(n, left_child, right_child)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[] $leftChild
* @param Integer[] $rightChild
* @return Boolean
*/
function validateBinaryTreeNodes($n, $leftChild, $rightChild) {

}
}
```

Dart Solution:

```
class Solution {
bool validateBinaryTreeNodes(int n, List<int> leftChild, List<int>
rightChild) {

}
}
```

Scala Solution:

```scala
object Solution {
def validateBinaryTreeNodes(n: Int, leftChild: Array[Int], rightChild:
Array[Int]): Boolean = {

}
}
```

Elixir Solution:

```elixir
defmodule Solution do
@spec validate_binary_tree_nodes(n :: integer, left_child :: [integer],
right_child :: [integer]) :: boolean
def validate_binary_tree_nodes(n, left_child, right_child) do

end
end
```

Erlang Solution:

```erlang
-spec validate_binary_tree_nodes(N :: integer(), LeftChild :: [integer()],
RightChild :: [integer()]) -> boolean().
validate_binary_tree_nodes(N, LeftChild, RightChild) ->
.
```

Racket Solution:

```racket
(define/contract (validate-binary-tree-nodes n leftChild rightChild)
(-> exact-integer? (listof exact-integer?) (listof exact-integer?) boolean?)
)
```