Problem 261: Graph Valid Tree

Problem Information

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

Problem Description

You have a graph of

n

nodes labeled from

0

to

n - 1

. You are given an integer n and a list of

edges

where

edges[i] = [a

i

, b

i

]

indicates that there is an undirected edge between nodes

a

i

and

b

i

in the graph.

Return

true

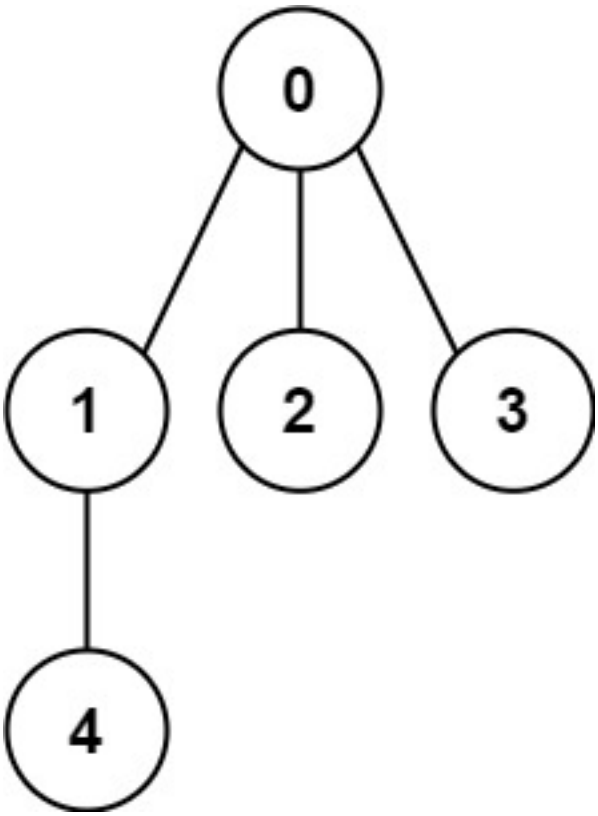
if the edges of the given graph make up a valid tree, and

false

otherwise

.

Example 1:



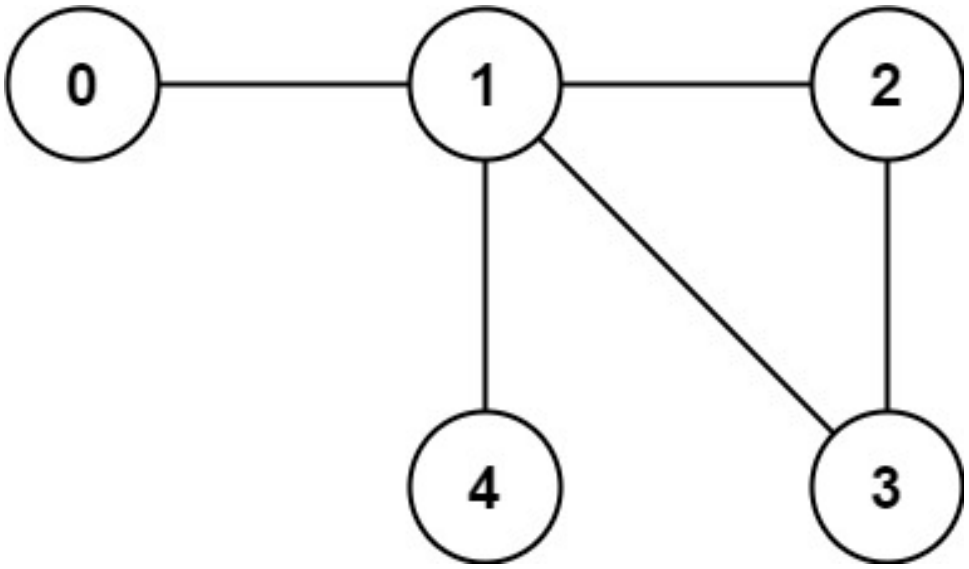
Input:

n = 5, edges = [[0,1],[0,2],[0,3],[1,4]]

Output:

true

Example 2:



Input:

n = 5, edges = [[0,1],[1,2],[2,3],[1,3],[1,4]]

Output:

false

Constraints:

1 <= n <= 2000

0 <= edges.length <= 5000

edges[i].length == 2

0 <= a

i

, b

i

< n

a

i

!= b

i

There are no self-loops or repeated edges.

Code Snippets

C++:

```
class Solution {
public:
bool validTree(int n, vector<vector<int>>& edges) {

}
};
```

Java:

```
class Solution {
public boolean validTree(int n, int[][] edges) {

}
}
```

Python3:

```
class Solution:
def validTree(self, n: int, edges: List[List[int]]) -> bool:
```

Python:

```
class Solution(object):
def validTree(self, n, edges):
"""
:type n: int
```

```
:type edges: List[List[int]]
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} edges
* @return {boolean}
*/
var validTree = function(n, edges) {

};
```

TypeScript:

```
function validTree(n: number, edges: number[][]): boolean {

};
```

C#:

```
public class Solution {
public bool ValidTree(int n, int[][] edges) {

}
}
```

C:

```
bool validTree(int n, int** edges, int edgesSize, int* edgesColSize) {

}
```

Go:

```
func validTree(n int, edges [][]int) bool {

}
```

Kotlin:

```
class Solution {
fun validTree(n: Int, edges: Array<IntArray>): Boolean {

}
}
```

Swift:

```
class Solution {
func validTree(_ n: Int, _ edges: [[Int]]) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn valid_tree(n: i32, edges: Vec<Vec<i32>>) -> bool {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} edges
# @return {Boolean}
def valid_tree(n, edges)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edges
* @return Boolean
*/
function validTree($n, $edges) {

}
```

```
}
```

Dart:

```
class Solution {
bool validTree(int n, List<List<int>> edges) {

}
}
```

Scala:

```
object Solution {
def validTree(n: Int, edges: Array[Array[Int]]): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec valid_tree(n :: integer, edges :: [[integer]]) :: boolean
def valid_tree(n, edges) do

end
end
```

Erlang:

```
-spec valid_tree(N :: integer(), Edges :: [[integer()]]) -> boolean().
valid_tree(N, Edges) ->
.
```

Racket:

```
(define/contract (valid-tree n edges)
(-> exact-integer? (listof (listof exact-integer?)) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Graph Valid Tree
* 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 validTree(int n, vector<vector<int>>& edges) {

}
};
```

Java Solution:

```
/**
* Problem: Graph Valid Tree
* 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 validTree(int n, int[][] edges) {

}
}
```

Python3 Solution:

```
"""
Problem: Graph Valid Tree
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 validTree(self, n: int, edges: List[List[int]]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def validTree(self, n, edges):
"""
:type n: int
:type edges: List[List[int]]
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Graph Valid Tree
* 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[][]} edges
* @return {boolean}
*/
var validTree = function(n, edges) {

};
```

TypeScript Solution:

```
/**
* Problem: Graph Valid Tree
* 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 validTree(n: number, edges: number[][]): boolean {

};
```

C# Solution:

```
/*
* Problem: Graph Valid Tree
* 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 ValidTree(int n, int[][] edges) {

}
}
```

C Solution:

```
/*
* Problem: Graph Valid Tree
* 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 validTree(int n, int** edges, int edgesSize, int* edgesColSize) {

}
```

Go Solution:

```
// Problem: Graph Valid Tree
// 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 validTree(n int, edges [][]int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun validTree(n: Int, edges: Array<IntArray>): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func validTree(_ n: Int, _ edges: [[Int]]) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Graph Valid Tree
// 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 valid_tree(n: i32, edges: Vec<Vec<i32>>) -> bool {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} edges
# @return {Boolean}
def valid_tree(n, edges)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edges
* @return Boolean
*/
function validTree($n, $edges) {

}
}
```

Dart Solution:

```
class Solution {
bool validTree(int n, List<List<int>> edges) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def validTree(n: Int, edges: Array[Array[Int]]): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec valid_tree(n :: integer, edges :: [[integer]]) :: boolean
def valid_tree(n, edges) do

end
end
```

Erlang Solution:

```
-spec valid_tree(N :: integer(), Edges :: [[integer()]]) -> boolean().
valid_tree(N, Edges) ->
.
```

Racket Solution:

```
(define/contract (valid-tree n edges)
(-> exact-integer? (listof (listof exact-integer?)) boolean?)
)
```