Problem 3249: Count the Number of Good Nodes

Problem Information

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

Problem Description

There is an

undirected

tree with

n

nodes labeled from

0

to

n - 1

, and rooted at node

0

. You are given a 2D integer array

edges

of length

n - 1

, where

edges[i] = [a

i

, b

i

]

indicates that there is an edge between nodes

a

i

and

b

i

in the tree.

A node is

good

if all the

subtrees

rooted at its children have the same size.

Return the number of

good

nodes in the given tree.

A

subtree

of

treeName

is a tree consisting of a node in

treeName

and all of its descendants.

Example 1:

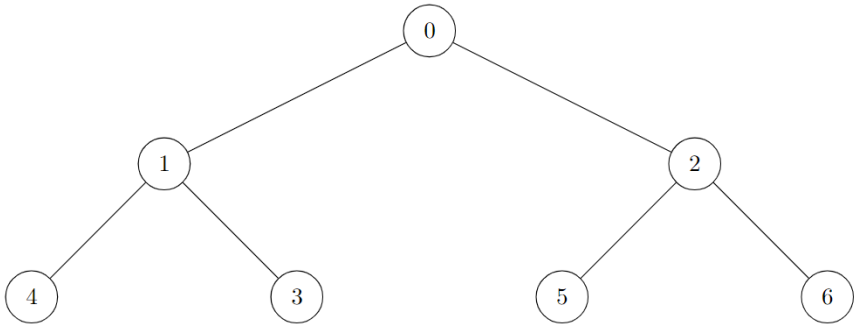
Input:

edges = [[0,1],[0,2],[1,3],[1,4],[2,5],[2,6]]

Output:

7

Explanation:



All of the nodes of the given tree are good.

Example 2:

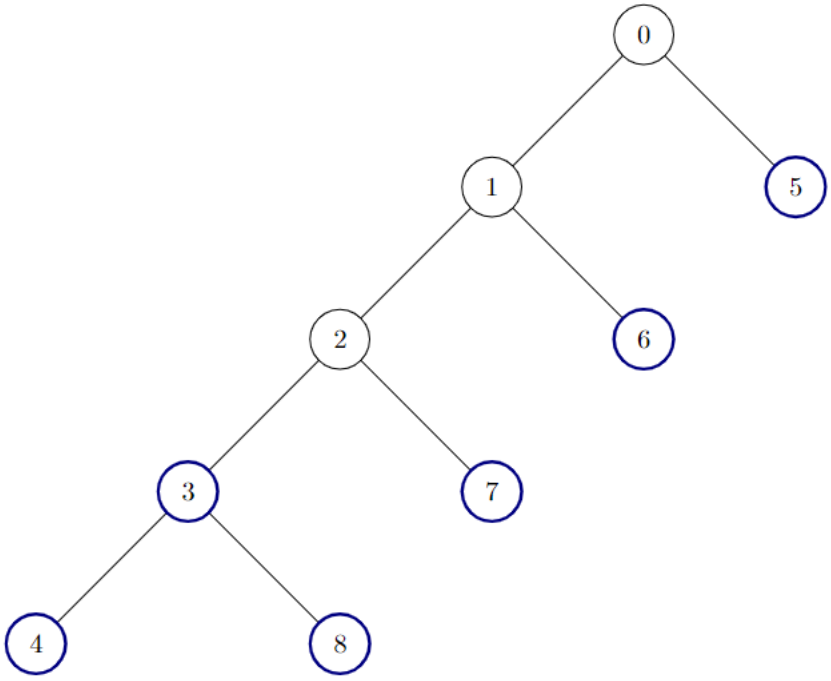
Input:

edges = [[0,1],[1,2],[2,3],[3,4],[0,5],[1,6],[2,7],[3,8]]

Output:

6

Explanation:



There are 6 good nodes in the given tree. They are colored in the image above.

Example 3:

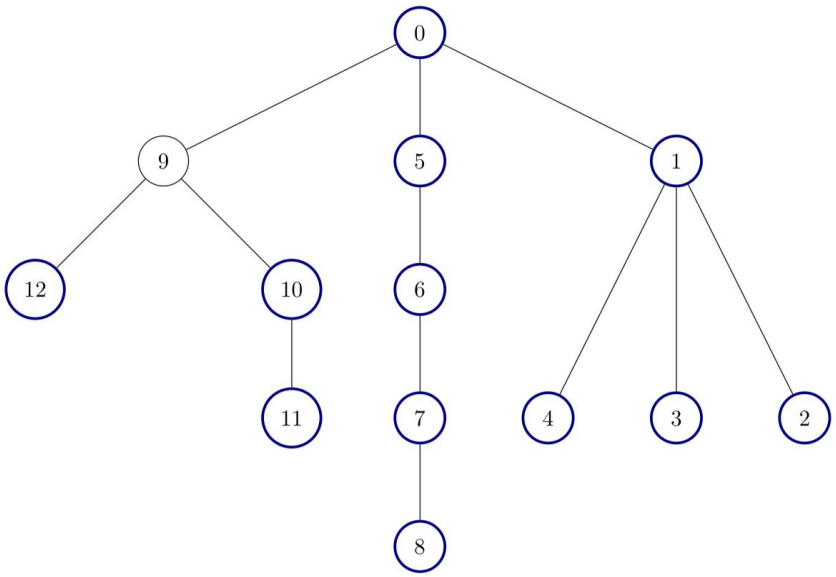
Input:

edges = [[0,1],[1,2],[1,3],[1,4],[0,5],[5,6],[6,7],[7,8],[0,9],[9,10],[9,12],[10,11]]

Output:

12

Explanation:



All nodes except node 9 are good.

Constraints:

2 <= n <= 10

5

edges.length == n - 1

edges[i].length == 2

0 <= a

i

, b

i

< n

The input is generated such that

edges

represents a valid tree.

Code Snippets

C++:

```
class Solution {
public:
int countGoodNodes(vector<vector<int>>& edges) {

}
};
```

Java:

```
class Solution {
public int countGoodNodes(int[][] edges) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[][]} edges
* @return {number}
*/
```

```
var countGoodNodes = function(edges) {

};
```

TypeScript:

```
function countGoodNodes(edges: number[][]): number {

};
```

C#:

```
public class Solution {
public int CountGoodNodes(int[][] edges) {

}
}
```

C:

```
int countGoodNodes(int** edges, int edgesSize, int* edgesColSize) {

}
```

Go:

```
func countGoodNodes(edges [][]int) int {

}
```

Kotlin:

```
class Solution {
fun countGoodNodes(edges: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func countGoodNodes(_ edges: [[Int]]) -> Int {
```

```
}
}
```

Rust:

```
impl Solution {
pub fn count_good_nodes(edges: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} edges
# @return {Integer}
def count_good_nodes(edges)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $edges
* @return Integer
*/
function countGoodNodes($edges) {

}
}
```

Dart:

```
class Solution {
int countGoodNodes(List<List<int>> edges) {

}
}
```

Scala:

```
object Solution {
def countGoodNodes(edges: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_good_nodes(edges :: [[integer]]) :: integer
def count_good_nodes(edges) do

end
end
```

Erlang:

```
-spec count_good_nodes(Edges :: [[integer()]]) -> integer().
count_good_nodes(Edges) ->
.
```

Racket:

```
(define/contract (count-good-nodes edges)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count the Number of Good Nodes
* Difficulty: Medium
* Tags: array, 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
*/
```

```
class Solution {
public:
int countGoodNodes(vector<vector<int>>& edges) {

}
};
```

Java Solution:

```
/**
* Problem: Count the Number of Good Nodes
* Difficulty: Medium
* Tags: array, 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
*/

class Solution {
public int countGoodNodes(int[][] edges) {

}
}
```

Python3 Solution:

```
"""
Problem: Count the Number of Good Nodes
Difficulty: Medium
Tags: array, 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
"""

class Solution:
def countGoodNodes(self, edges: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count the Number of Good Nodes
* Difficulty: Medium
* Tags: array, 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
*/

/**
* @param {number[][]} edges
* @return {number}
*/
var countGoodNodes = function(edges) {

};
```

TypeScript Solution:

```
/**
* Problem: Count the Number of Good Nodes
* Difficulty: Medium
* Tags: array, 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
*/

function countGoodNodes(edges: number[][]): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Count the Number of Good Nodes
* Difficulty: Medium
* Tags: array, 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
*/

public class Solution {
public int CountGoodNodes(int[][] edges) {

}
}
```

C Solution:

```
/*
* Problem: Count the Number of Good Nodes
* Difficulty: Medium
* Tags: array, 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
*/

int countGoodNodes(int** edges, int edgesSize, int* edgesColSize) {

}
```

Go Solution:

```
// Problem: Count the Number of Good Nodes
// Difficulty: Medium
```

```
// Tags: array, 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

func countGoodNodes(edges [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun countGoodNodes(edges: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func countGoodNodes(_ edges: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count the Number of Good Nodes
// Difficulty: Medium
// Tags: array, 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

impl Solution {
pub fn count_good_nodes(edges: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} edges
# @return {Integer}
def count_good_nodes(edges)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $edges
* @return Integer
*/
function countGoodNodes($edges) {

}
}
```

Dart Solution:

```
class Solution {
int countGoodNodes(List<List<int>> edges) {

}
}
```

Scala Solution:

```
object Solution {
def countGoodNodes(edges: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_good_nodes(edges :: [[integer]]) :: integer
def count_good_nodes(edges) do
```

```
end
end
```

Erlang Solution:

```
-spec count_good_nodes(Edges :: [[integer()]]) -> integer().
count_good_nodes(Edges) ->
.
```

Racket Solution:

```
(define/contract (count-good-nodes edges)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```