Problem 2581: Count Number of Possible Root Nodes

Problem Information

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

Problem Description

Alice has an undirected tree with

n

nodes labeled from

0

to

n - 1

. The tree is represented as 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.

Alice wants Bob to find the root of the tree. She allows Bob to make several

guesses

about her tree. In one guess, he does the following:

Chooses two

distinct

integers

u

and

v

such that there exists an edge

[u, v]

in the tree.

He tells Alice that

u

is the

parent

of

v

in the tree.

Bob's guesses are represented by a 2D integer array

guesses

where

guesses[j] = [u

j

, v

j

]

indicates Bob guessed

u

j

to be the parent of

v

j

.

Alice being lazy, does not reply to each of Bob's guesses, but just says that

at least

k

of his guesses are

true

.

Given the 2D integer arrays

edges

,

guesses

and the integer

k

, return

the

number of possible nodes

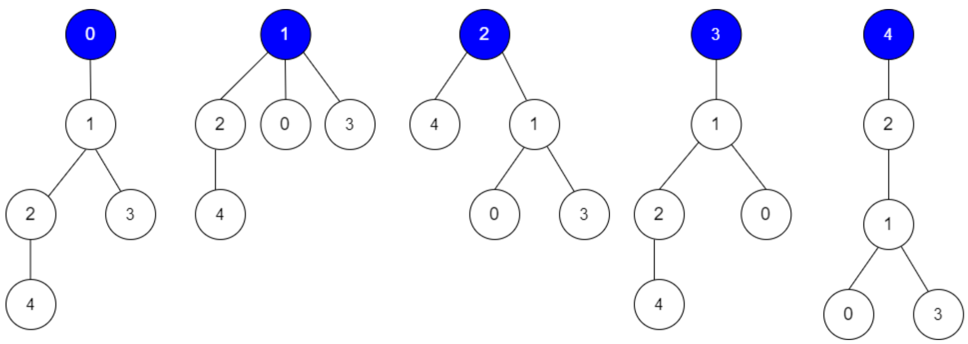
that can be the root of Alice's tree

. If there is no such tree, return

0

.

Example 1:



Input:

edges = [[0,1],[1,2],[1,3],[4,2]], guesses = [[1,3],[0,1],[1,0],[2,4]], k = 3

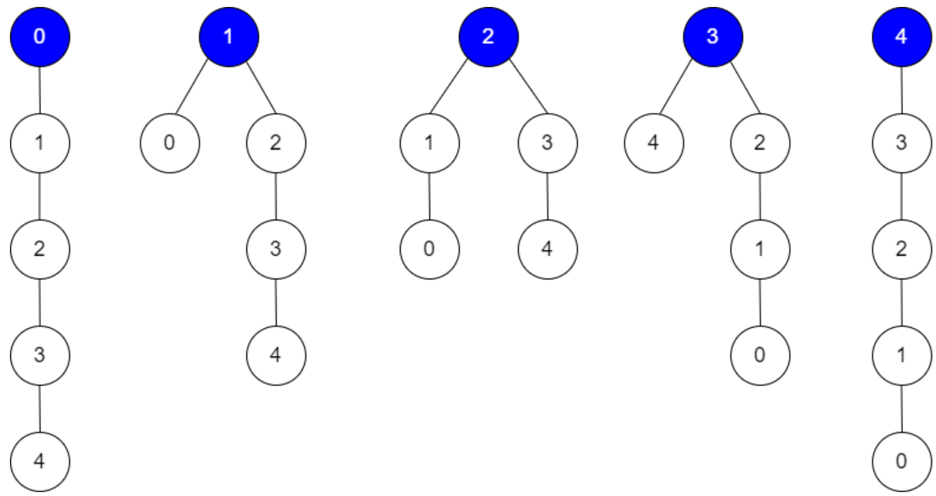
Output:

3

Explanation:

Root = 0, correct guesses = [1,3], [0,1], [2,4] Root = 1, correct guesses = [1,3], [1,0], [2,4] Root = 2, correct guesses = [1,3], [1,0], [2,4] Root = 3, correct guesses = [1,0], [2,4] Root = 4, correct guesses = [1,3], [1,0] Considering 0, 1, or 2 as root node leads to 3 correct guesses.

Example 2:



Input:

edges = [[0,1],[1,2],[2,3],[3,4]], guesses = [[1,0],[3,4],[2,1],[3,2]], k = 1

Output:

5

Explanation:

Root = 0, correct guesses = [3,4] Root = 1, correct guesses = [1,0], [3,4] Root = 2, correct guesses = [1,0], [2,1], [3,4] Root = 3, correct guesses = [1,0], [2,1], [3,2], [3,4] Root = 4, correct guesses = [1,0], [2,1], [3,2] Considering any node as root will give at least 1 correct guess.

Constraints:

edges.length == n - 1

2 <= n <= 10

5

1 <= guesses.length <= 10

5

0 <= a

i

, b

i

, u

j

, v

j

<= n - 1

a

i

!= b

i

u

j

!= v

j

edges

represents a valid tree.

guesses[j]

is an edge of the tree.

guesses

is unique.

0 <= k <= guesses.length

Code Snippets

C++:

```
class Solution {
public:
int rootCount(vector<vector<int>>& edges, vector<vector<int>>& guesses, int
k) {

}
};
```

Java:

```
class Solution {
public int rootCount(int[][] edges, int[][] guesses, int k) {

}
}
```

Python3:

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

Python:

```
class Solution(object):
def rootCount(self, edges, guesses, k):
"""
:type edges: List[List[int]]
:type guesses: List[List[int]]
```

```
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} edges
* @param {number[][]} guesses
* @param {number} k
* @return {number}
*/
var rootCount = function(edges, guesses, k) {

};
```

TypeScript:

```
function rootCount(edges: number[][], guesses: number[][], k: number): number
{

};
```

C#:

```
public class Solution {
public int RootCount(int[][] edges, int[][] guesses, int k) {

}
}
```

C:

```
int rootCount(int** edges, int edgesSize, int* edgesColSize, int** guesses,
int guessesSize, int* guessesColSize, int k) {

}
```

Go:

```
func rootCount(edges [][]int, guesses [][]int, k int) int {
```

```
}
```

Kotlin:

```
class Solution {
fun rootCount(edges: Array<IntArray>, guesses: Array<IntArray>, k: Int): Int
{

}
}
```

Swift:

```
class Solution {
func rootCount(_ edges: [[Int]], _ guesses: [[Int]], _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn root_count(edges: Vec<Vec<i32>>, guesses: Vec<Vec<i32>>, k: i32) ->
i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} edges
# @param {Integer[][]} guesses
# @param {Integer} k
# @return {Integer}
def root_count(edges, guesses, k)

end
```

PHP:

```
class Solution {
```

```
/**
* @param Integer[][] $edges
* @param Integer[][] $guesses
* @param Integer $k
* @return Integer
*/
function rootCount($edges, $guesses, $k) {

}
}
```

Dart:

```
class Solution {
int rootCount(List<List<int>> edges, List<List<int>> guesses, int k) {

}
}
```

Scala:

```
object Solution {
def rootCount(edges: Array[Array[Int]], guesses: Array[Array[Int]], k: Int):
Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec root_count(edges :: [[integer]], guesses :: [[integer]], k :: integer)
:: integer
def root_count(edges, guesses, k) do

end
end
```

Erlang:

```
-spec root_count(Edges :: [[integer()]], Guesses :: [[integer()]], K ::
integer()) -> integer().
```

```
root_count(Edges, Guesses, K) ->
.
```

Racket:

```
(define/contract (root-count edges guesses k)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?))
exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Number of Possible Root Nodes
* Difficulty: Hard
* Tags: array, tree, dp, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int rootCount(vector<vector<int>>& edges, vector<vector<int>>& guesses, int
k) {

}
};
```

Java Solution:

```
/**
* Problem: Count Number of Possible Root Nodes
* Difficulty: Hard
* Tags: array, tree, dp, hash, search
*
* Approach: Use two pointers or sliding window technique
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int rootCount(int[][] edges, int[][] guesses, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Number of Possible Root Nodes
Difficulty: Hard
Tags: array, tree, dp, hash, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Number of Possible Root Nodes
* Difficulty: Hard
* Tags: array, tree, dp, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[][]} edges
* @param {number[][]} guesses
* @param {number} k
* @return {number}
*/
var rootCount = function(edges, guesses, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Number of Possible Root Nodes
* Difficulty: Hard
* Tags: array, tree, dp, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function rootCount(edges: number[][], guesses: number[][], k: number): number
{

};
```

C# Solution:

```
/*
* Problem: Count Number of Possible Root Nodes
* Difficulty: Hard
```

```
* Tags: array, tree, dp, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int RootCount(int[][] edges, int[][] guesses, int k) {

}
}
```

C Solution:

```
/*
* Problem: Count Number of Possible Root Nodes
* Difficulty: Hard
* Tags: array, tree, dp, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int rootCount(int** edges, int edgesSize, int* edgesColSize, int** guesses,
int guessesSize, int* guessesColSize, int k) {

}
```

Go Solution:

```
// Problem: Count Number of Possible Root Nodes
// Difficulty: Hard
// Tags: array, tree, dp, hash, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func rootCount(edges [][]int, guesses [][]int, k int) int {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun rootCount(edges: Array<IntArray>, guesses: Array<IntArray>, k: Int): Int
{

}
}
```

Swift Solution:

```
class Solution {
func rootCount(_ edges: [[Int]], _ guesses: [[Int]], _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count Number of Possible Root Nodes
// Difficulty: Hard
// Tags: array, tree, dp, hash, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn root_count(edges: Vec<Vec<i32>>, guesses: Vec<Vec<i32>>, k: i32) ->
i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} edges
# @param {Integer[][]} guesses
```

```
# @param {Integer} k
# @return {Integer}
def root_count(edges, guesses, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $edges
* @param Integer[][] $guesses
* @param Integer $k
* @return Integer
*/
function rootCount($edges, $guesses, $k) {

}
}
```

Dart Solution:

```
class Solution {
int rootCount(List<List<int>> edges, List<List<int>> guesses, int k) {

}
}
```

Scala Solution:

```
object Solution {
def rootCount(edges: Array[Array[Int]], guesses: Array[Array[Int]], k: Int):
Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec root_count(edges :: [[integer]], guesses :: [[integer]], k :: integer)
:: integer
def root_count(edges, guesses, k) do

end
end
```

Erlang Solution:

```
-spec root_count(Edges :: [[integer()]], Guesses :: [[integer()]], K ::
integer()) -> integer().
root_count(Edges, Guesses, K) ->
.
```

Racket Solution:

```
(define/contract (root-count edges guesses k)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?))
exact-integer? exact-integer?)
)
```