Problem 3715: Sum of Perfect Square Ancestors

Problem Information

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

Problem Description

You are given an integer

n

and an undirected tree rooted at node 0 with

n

nodes numbered from 0 to

n - 1

. This is represented by a 2D array

edges

of length

n - 1

, where

edges[i] = [u

i

, v

i

]

indicates an undirected edge between nodes

u

i

and

v

i

.

You are also given an integer array

nums

, where

nums[i]

is the positive integer assigned to node

i

.

Define a value

t

i

as the number of

ancestors

of node

i

such that the product

nums[i] * nums[ancestor]

is a

perfect square

.

Return the sum of all

t

i

values for all nodes

i

in range

[1, n - 1]

.

Note

:

In a rooted tree, the

ancestors

of node

i

are all nodes on the path from node

i

to the root node 0,

excluding

i

itself.

Example 1:

Input:

n = 3, edges = [[0,1],[1,2]], nums = [2,8,2]

Output:

3

Explanation:

i

Ancestors

nums[i] * nums[ancestor]

Square Check

t

i

1

[0]

nums[1] * nums[0] = 8 * 2 = 16

16 is a perfect square

1

2

[1, 0]

nums[2] * nums[1] = 2 * 8 = 16

nums[2] * nums[0] = 2 * 2 = 4

Both 4 and 16 are perfect squares

2

Thus, the total number of valid ancestor pairs across all non-root nodes is

1 + 2 = 3

.

Example 2:

Input:

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

Output:

1

Explanation:

i

Ancestors

nums[i] * nums[ancestor]

Square Check

t

i

1

[0]

nums[1] * nums[0] = 2 * 1 = 2

2 is

not

a perfect square

0

2

[0]

nums[2] * nums[0] = 4 * 1 = 4

4 is a perfect square

1

Thus, the total number of valid ancestor pairs across all non-root nodes is 1.

Example 3:

Input:

n = 4, edges = [[0,1],[0,2],[1,3]], nums = [1,2,9,4]

Output:

2

Explanation:

i

Ancestors

nums[i] * nums[ancestor]

Square Check

t

i

1

[0]

nums[1] * nums[0] = 2 * 1 = 2

2 is

not

a perfect square

0

2

[0]

nums[2] * nums[0] = 9 * 1 = 9

9 is a perfect square

1

3

[1, 0]

nums[3] * nums[1] = 4 * 2 = 8

nums[3] * nums[0] = 4 * 1 = 4

Only 4 is a perfect square

1

Thus, the total number of valid ancestor pairs across all non-root nodes is

0 + 1 + 1 = 2

.

Constraints:

1 <= n <= 10

5

edges.length == n - 1

edges[i] = [u

i

, v

i

]

0 <= u

i

, v

i

<= n - 1

nums.length == n

1 <= nums[i] <= 10

5

The input is generated such that

edges

represents a valid tree.

Code Snippets

C++:

```
class Solution {
public:
long long sumOfAncestors(int n, vector<vector<int>>& edges, vector<int>&
nums) {

}
};
```

Java:

```
class Solution {
public long sumOfAncestors(int n, int[][] edges, int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

```
};
```

TypeScript:

```
function sumOfAncestors(n: number, edges: number[][], nums: number[]): number
{

};
```

C#:

```
public class Solution {
public long SumOfAncestors(int n, int[][] edges, int[] nums) {

}
}
```

C:

```
long long sumOfAncestors(int n, int** edges, int edgesSize, int*
edgesColSize, int* nums, int numsSize) {

}
```

Go:

```
func sumOfAncestors(n int, edges [][]int, nums []int) int64 {

}
```

Kotlin:

```
class Solution {
fun sumOfAncestors(n: Int, edges: Array<IntArray>, nums: IntArray): Long {

}
}
```

Swift:

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

```
}
}
```

Rust:

```
impl Solution {
pub fn sum_of_ancestors(n: i32, edges: Vec<Vec<i32>>, nums: Vec<i32>) -> i64
{

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
int sumOfAncestors(int n, List<List<int>> edges, List<int> nums) {
```

```
}
}
```

Scala:

```
object Solution {
def sumOfAncestors(n: Int, edges: Array[Array[Int]], nums: Array[Int]): Long
= {

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec sum_of_ancestors(N :: integer(), Edges :: [[integer()]], Nums ::
[integer()]) -> integer().
sum_of_ancestors(N, Edges, Nums) ->
.
```

Racket:

```
(define/contract (sum-of-ancestors n edges nums)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?)
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sum of Perfect Square Ancestors
* Difficulty: Hard
* Tags: array, tree, math, hash, 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:
long long sumOfAncestors(int n, vector<vector<int>>& edges, vector<int>&
nums) {

}
};
```

Java Solution:

```
/**
* Problem: Sum of Perfect Square Ancestors
* Difficulty: Hard
* Tags: array, tree, math, hash, 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 long sumOfAncestors(int n, int[][] edges, int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Sum of Perfect Square Ancestors
Difficulty: Hard
Tags: array, tree, math, hash, 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 sumOfAncestors(self, n: int, edges: List[List[int]], nums: List[int]) ->
int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Sum of Perfect Square Ancestors
* Difficulty: Hard
* Tags: array, tree, math, hash, 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} n
* @param {number[][]} edges
* @param {number[]} nums
* @return {number}
*/
```

```
var sumOfAncestors = function(n, edges, nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Sum of Perfect Square Ancestors
* Difficulty: Hard
* Tags: array, tree, math, hash, 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 sumOfAncestors(n: number, edges: number[][], nums: number[]): number
{

};
```

C# Solution:

```
/*
* Problem: Sum of Perfect Square Ancestors
* Difficulty: Hard
* Tags: array, tree, math, hash, 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 long SumOfAncestors(int n, int[][] edges, int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Sum of Perfect Square Ancestors
* Difficulty: Hard
* Tags: array, tree, math, hash, 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
*/

long long sumOfAncestors(int n, int** edges, int edgesSize, int*
edgesColSize, int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Sum of Perfect Square Ancestors
// Difficulty: Hard
// Tags: array, tree, math, hash, 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 sumOfAncestors(n int, edges [][]int, nums []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun sumOfAncestors(n: Int, edges: Array<IntArray>, nums: IntArray): Long {

}
}
```

Swift Solution:

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

```
}
}
```

Rust Solution:

```
// Problem: Sum of Perfect Square Ancestors
// Difficulty: Hard
// Tags: array, tree, math, hash, 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 sum_of_ancestors(n: i32, edges: Vec<Vec<i32>>, nums: Vec<i32>) -> i64
{

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edges
* @param Integer[] $nums
* @return Integer
*/
function sumOfAncestors($n, $edges, $nums) {
```

```
}
}
```

Dart Solution:

```
class Solution {
int sumOfAncestors(int n, List<List<int>> edges, List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def sumOfAncestors(n: Int, edges: Array[Array[Int]], nums: Array[Int]): Long
= {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec sum_of_ancestors(N :: integer(), Edges :: [[integer()]], Nums ::
[integer()]) -> integer().
sum_of_ancestors(N, Edges, Nums) ->
.
```

Racket Solution:

```
(define/contract (sum-of-ancestors n edges nums)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?)
exact-integer?)
)
```