Problem 1766: Tree of Coprimes

Problem Information

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

Problem Description

There is a tree (i.e., a connected, undirected graph that has no cycles) consisting of

n

nodes numbered from

0

to

n - 1

and exactly

n - 1

edges. Each node has a value associated with it, and the

root

of the tree is node

0

.

To represent this tree, you are given an integer array

nums

and a 2D array

edges

. Each

nums[i]

represents the

i

th

node's value, and each

edges[j] = [u

j

, v

j

]

represents an edge between nodes

u

j

and

v

j

in the tree.

Two values

x

and

y

are

coprime

if

gcd(x, y) == 1

where

gcd(x, y)

is the

greatest common divisor

of

x

and

y

.

An ancestor of a node

i

is any other node on the shortest path from node

i

to the

root

. A node is

not

considered an ancestor of itself.

Return

an array

ans

of size

n

,

where

$ans[i]$

is the closest ancestor to node

i

such that

nums[i]

and

nums[ans[i]]

are

coprime

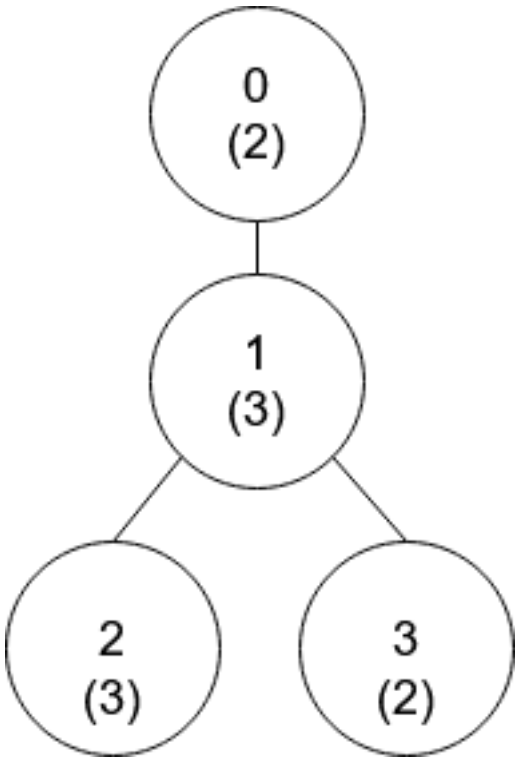
, or

-1

if there is no such ancestor

.

Example 1:



Input:

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

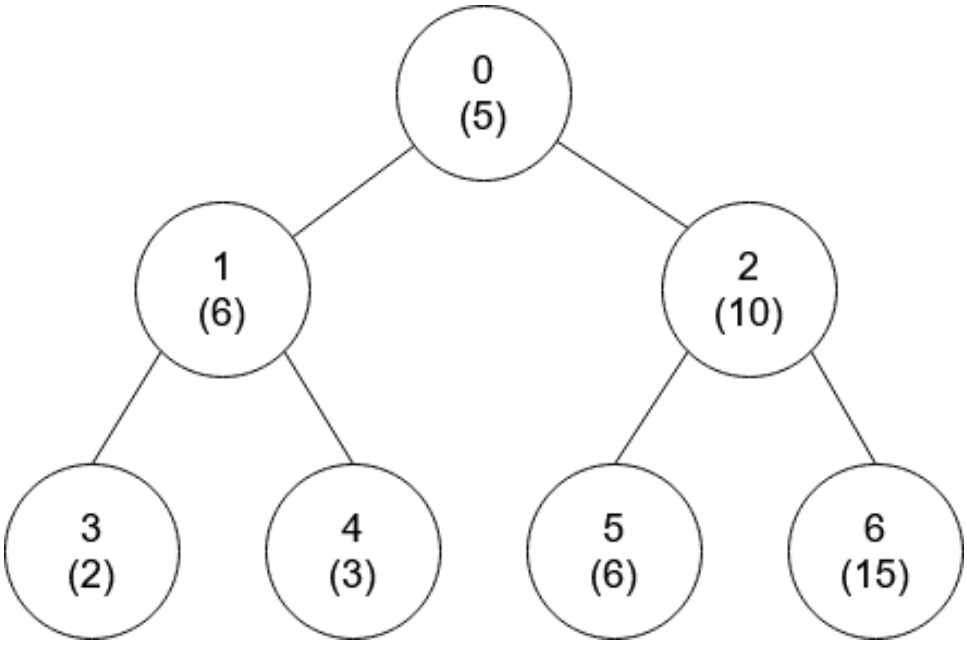
Output:

[-1,0,0,1]

Explanation:

In the above figure, each node's value is in parentheses. - Node 0 has no coprime ancestors. - Node 1 has only one ancestor, node 0. Their values are coprime (gcd(2,3) == 1). - Node 2 has two ancestors, nodes 1 and 0. Node 1's value is not coprime (gcd(3,3) == 3), but node 0's value is (gcd(2,3) == 1), so node 0 is the closest valid ancestor. - Node 3 has two ancestors, nodes 1 and 0. It is coprime with node 1 (gcd(3,2) == 1), so node 1 is its closest valid ancestor.

Example 2:



Input:

nums = [5,6,10,2,3,6,15], edges = [[0,1],[0,2],[1,3],[1,4],[2,5],[2,6]]

Output:

[-1,0,-1,0,0,0,-1]

Constraints:

nums.length == n

1 <= nums[i] <= 50

1 <= n <= 10

5

edges.length == n - 1

edges[j].length == 2

0 <= u

j

, v

j

< n

u

j

!= v

j

Code Snippets

C++:

```
class Solution {
public:
```

```
vector<int> getCoprimes(vector<int>& nums, vector<vector<int>>& edges) {

}
};
```

Java:

```
class Solution {
public int[] getCoprimes(int[] nums, int[][] edges) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public int[] GetCoprimes(int[] nums, int[][] edges) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* getCoprimes(int* nums, int numsSize, int** edges, int edgesSize, int*
edgesColSize, int* returnSize) {

}
```

Go:

```
func getCoprimes(nums []int, edges [][]int) []int {

}
```

Kotlin:

```
class Solution {
fun getCoprimes(nums: IntArray, edges: Array<IntArray>): IntArray {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

```
}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (get-coprimes nums edges)
(-> (listof exact-integer?) (listof (listof exact-integer?)) (listof
exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Tree of Coprimes
* Difficulty: Hard
* Tags: array, tree, graph, math, 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:
vector<int> getCoprimes(vector<int>& nums, vector<vector<int>>& edges) {

}
};
```

Java Solution:

```
/**
* Problem: Tree of Coprimes
* Difficulty: Hard
* Tags: array, tree, graph, math, 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[] getCoprimes(int[] nums, int[][] edges) {

}
}
```

Python3 Solution:

```
"""
Problem: Tree of Coprimes
Difficulty: Hard
Tags: array, tree, graph, math, 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 getCoprimes(self, nums: List[int], edges: List[List[int]]) -> List[int]:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Tree of Coprimes
* Difficulty: Hard
* Tags: array, tree, graph, math, 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[]} nums
* @param {number[][]} edges
* @return {number[]}
*/
var getCoprimes = function(nums, edges) {

};
```

TypeScript Solution:

```
/**
* Problem: Tree of Coprimes
* Difficulty: Hard
* Tags: array, tree, graph, math, 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 getCoprimes(nums: number[], edges: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Tree of Coprimes
* Difficulty: Hard
* Tags: array, tree, graph, math, 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[] GetCoprimes(int[] nums, int[][] edges) {

}
}
```

C Solution:

```
/*
* Problem: Tree of Coprimes
* Difficulty: Hard
* Tags: array, tree, graph, math, 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
```

```
int* getCoprimes(int* nums, int numsSize, int** edges, int edgesSize, int*
edgesColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Tree of Coprimes
// Difficulty: Hard
// Tags: array, tree, graph, math, 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 getCoprimes(nums []int, edges [][]int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun getCoprimes(nums: IntArray, edges: Array<IntArray>): IntArray {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Tree of Coprimes
// Difficulty: Hard
// Tags: array, tree, graph, math, 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 get_coprimes(nums: Vec<i32>, edges: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (get-coprimes nums edges)
(-> (listof exact-integer?) (listof (listof exact-integer?)) (listof
exact-integer?))
)
```