Problem 2973: Find Number of Coins to Place in Tree Nodes

Problem Information

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

Problem Description

You are given 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.

You are also given a

0-indexed

integer array

cost

of length

n

, where

cost[i]

is the

cost

assigned to the

i

th

node.

You need to place some coins on every node of the tree. The number of coins to be placed at node

i

can be calculated as:

If size of the subtree of node

i

is less than

3

, place

1

coin.

Otherwise, place an amount of coins equal to the

maximum

product of cost values assigned to

3

distinct nodes in the subtree of node

i

. If this product is

negative

, place

0

coins.

Return

an array

coin

of size

n

such that

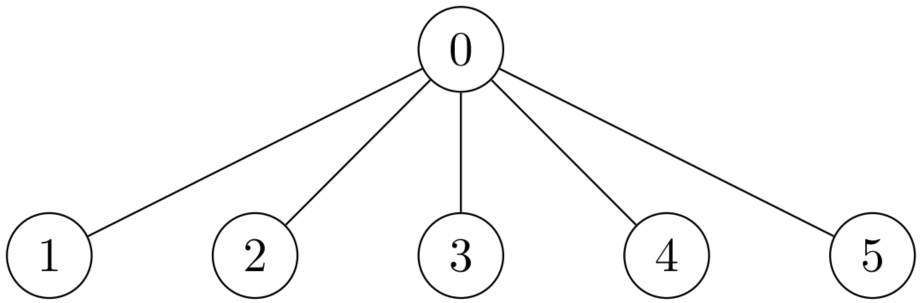
coin[i]

is the number of coins placed at node

i

.

Example 1:



Input:

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

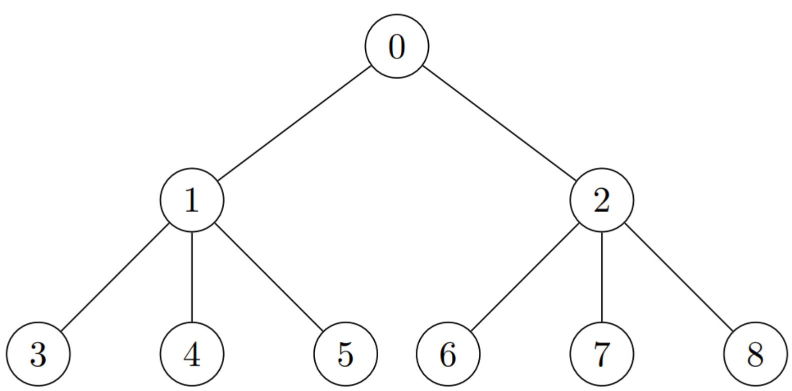
Output:

[120,1,1,1,1,1]

Explanation:

For node 0 place 6 * 5 * 4 = 120 coins. All other nodes are leaves with subtree of size 1, place 1 coin on each of them.

Example 2:



Input:

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

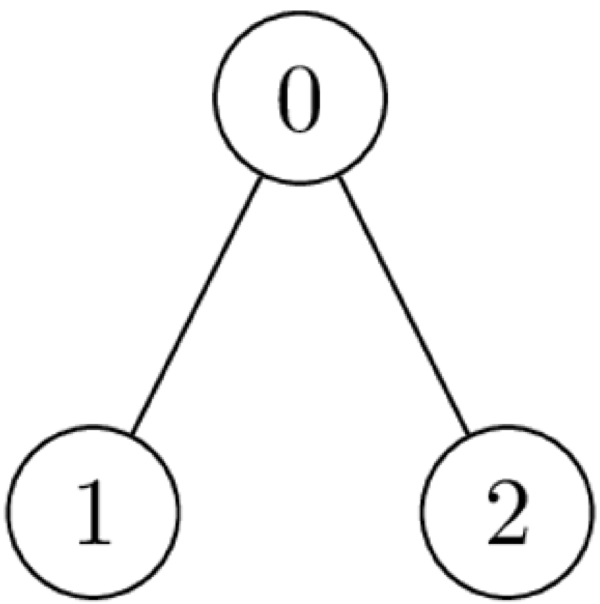
Output:

[280,140,32,1,1,1,1,1,1]

Explanation:

The coins placed on each node are: - Place 8 * 7 * 5 = 280 coins on node 0. - Place 7 * 5 * 4 = 140 coins on node 1. - Place 8 * 2 * 2 = 32 coins on node 2. - All other nodes are leaves with subtree of size 1, place 1 coin on each of them.

Example 3:



Input:

edges = [[0,1],[0,2]], cost = [1,2,-2]

Output:

[0,1,1]

Explanation:

Node 1 and 2 are leaves with subtree of size 1, place 1 coin on each of them. For node 0 the only possible product of cost is 2 * 1 * -2 = -4. Hence place 0 coins on node 0.

Constraints:

2 <= n <= 2 * 10

4

edges.length == n - 1

edges[i].length == 2

0 <= a

i

, b

i

< n

cost.length == n

1 <= |cost[i]| <= 10

4

The input is generated such that

edges

represents a valid tree.

Code Snippets

C++:

```
class Solution {
public:
vector<long long> placedCoins(vector<vector<int>>& edges, vector<int>& cost)
{

}
};
```

Java:

```
class Solution {
public long[] placedCoins(int[][] edges, int[] cost) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

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

};
```

TypeScript:

```
function placedCoins(edges: number[][], cost: number[]): number[] {

};
```

C#:

```
public class Solution {
public long[] PlacedCoins(int[][] edges, int[] cost) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
long long* placedCoins(int** edges, int edgesSize, int* edgesColSize, int*
cost, int costSize, int* returnSize) {

}
```

Go:

```
func placedCoins(edges [][]int, cost []int) []int64 {

}
```

Kotlin:

```
class Solution {
fun placedCoins(edges: Array<IntArray>, cost: IntArray): LongArray {
```

```
}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn placed_coins(edges: Vec<Vec<i32>>, cost: Vec<i32>) -> Vec<i64> {

}
}
```

Ruby:

```
# @param {Integer[][]} edges
# @param {Integer[]} cost
# @return {Integer[]}
def placed_coins(edges, cost)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $edges
* @param Integer[] $cost
* @return Integer[]
*/
function placedCoins($edges, $cost) {

}
}
```

Dart:

```
class Solution {
List<int> placedCoins(List<List<int>> edges, List<int> cost) {

}
}
```

Scala:

```
object Solution {
def placedCoins(edges: Array[Array[Int]], cost: Array[Int]): Array[Long] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec placed_coins(edges :: [[integer]], cost :: [integer]) :: [integer]
def placed_coins(edges, cost) do

end
end
```

Erlang:

```
-spec placed_coins(Edges :: [[integer()]], Cost :: [integer()]) ->
[integer()].
placed_coins(Edges, Cost) ->
.
```

Racket:

```
(define/contract (placed-coins edges cost)
(-> (listof (listof exact-integer?)) (listof exact-integer?) (listof
exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find Number of Coins to Place in Tree Nodes
* Difficulty: Hard
* Tags: array, tree, dp, sort, search, queue, heap
*
* 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:
vector<long long> placedCoins(vector<vector<int>>& edges, vector<int>& cost)
{

}
};
```

Java Solution:

```
/**
* Problem: Find Number of Coins to Place in Tree Nodes
* Difficulty: Hard
* Tags: array, tree, dp, sort, search, queue, heap
*
* 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 long[] placedCoins(int[][] edges, int[] cost) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Number of Coins to Place in Tree Nodes
Difficulty: Hard
```

```
Tags: array, tree, dp, sort, search, queue, heap

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 placedCoins(self, edges: List[List[int]], cost: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find Number of Coins to Place in Tree Nodes
* Difficulty: Hard
* Tags: array, tree, dp, sort, search, queue, heap
*
* 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[]} cost
* @return {number[]}
*/
var placedCoins = function(edges, cost) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Find Number of Coins to Place in Tree Nodes
* Difficulty: Hard
* Tags: array, tree, dp, sort, search, queue, heap
*
* 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 placedCoins(edges: number[][], cost: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Find Number of Coins to Place in Tree Nodes
* Difficulty: Hard
* Tags: array, tree, dp, sort, search, queue, heap
*
* 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 long[] PlacedCoins(int[][] edges, int[] cost) {

}
}
```

C Solution:

```
/*
* Problem: Find Number of Coins to Place in Tree Nodes
* Difficulty: Hard
```

```
* Tags: array, tree, dp, sort, search, queue, heap
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
long long* placedCoins(int** edges, int edgesSize, int* edgesColSize, int*
cost, int costSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Find Number of Coins to Place in Tree Nodes
// Difficulty: Hard
// Tags: array, tree, dp, sort, search, queue, heap
//
// 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 placedCoins(edges [][]int, cost []int) []int64 {

}
```

Kotlin Solution:

```
class Solution {
fun placedCoins(edges: Array<IntArray>, cost: IntArray): LongArray {

}
}
```

Swift Solution:

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

```
}
}
```

Rust Solution:

```
// Problem: Find Number of Coins to Place in Tree Nodes
// Difficulty: Hard
// Tags: array, tree, dp, sort, search, queue, heap
//
// 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 placed_coins(edges: Vec<Vec<i32>>, cost: Vec<i32>) -> Vec<i64> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} edges
# @param {Integer[]} cost
# @return {Integer[]}
def placed_coins(edges, cost)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $edges
* @param Integer[] $cost
* @return Integer[]
*/
function placedCoins($edges, $cost) {

}
```

```
}
```

Dart Solution:

```
class Solution {
List<int> placedCoins(List<List<int>> edges, List<int> cost) {

}
}
```

Scala Solution:

```
object Solution {
def placedCoins(edges: Array[Array[Int]], cost: Array[Int]): Array[Long] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec placed_coins(edges :: [[integer]], cost :: [integer]) :: [integer]
def placed_coins(edges, cost) do

end
end
```

Erlang Solution:

```
-spec placed_coins(Edges :: [[integer()]], Cost :: [integer()]) ->
[integer()].
placed_coins(Edges, Cost) ->
.
```

Racket Solution:

```
(define/contract (placed-coins edges cost)
(-> (listof (listof exact-integer?)) (listof exact-integer?) (listof
exact-integer?))
)
```