Problem 2603: Collect Coins in a Tree

Problem Information

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

Problem Description

There exists an undirected and unrooted tree with

n

nodes indexed from

0

to

n - 1

. You are given an integer

n

and 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 an array

coins

of size

n

where

coins[i]

can be either

0

or

1

, where

1

indicates the presence of a coin in the vertex

i

.

Initially, you choose to start at any vertex in the tree. Then, you can perform the following operations any number of times:

Collect all the coins that are at a distance of at most

2

from the current vertex, or

Move to any adjacent vertex in the tree.

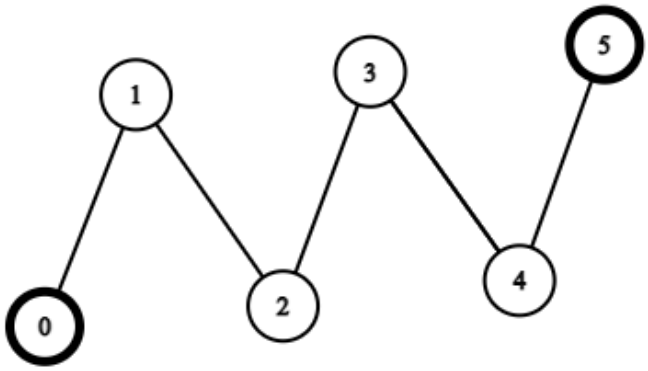
Find

the minimum number of edges you need to go through to collect all the coins and go back to the initial vertex

.

Note that if you pass an edge several times, you need to count it into the answer several times.

Example 1:



Input:

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

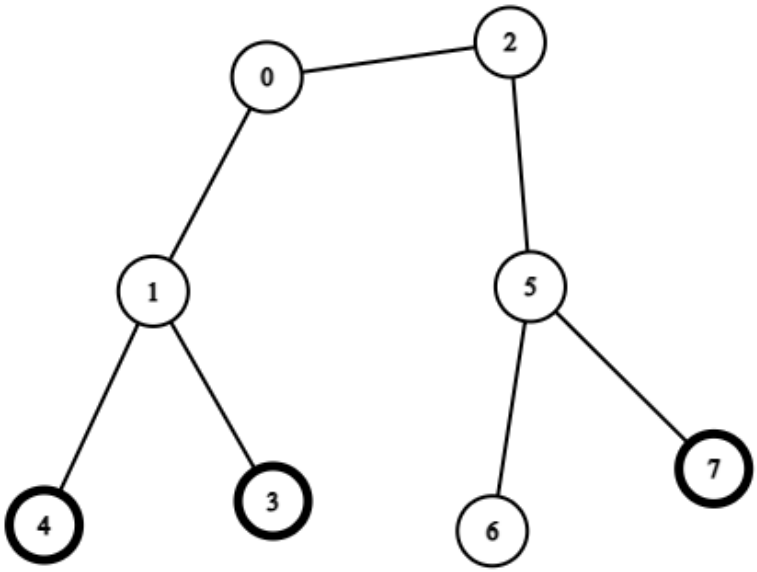
Output:

2

Explanation:

Start at vertex 2, collect the coin at vertex 0, move to vertex 3, collect the coin at vertex 5 then move back to vertex 2.

Example 2:



Input:

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

Output:

2

Explanation:

Start at vertex 0, collect the coins at vertices 4 and 3, move to vertex 2, collect the coin at vertex 7, then move back to vertex 0.

Constraints:

n == coins.length

1 <= n <= 3 * 10

4

0 <= coins[i] <= 1

edges.length == n - 1

edges[i].length == 2

0 <= a

i

, b

i

< n

a

i

!= b

i

edges

represents a valid tree.

Code Snippets

C++:

```
class Solution {
public:
int collectTheCoins(vector<int>& coins, vector<vector<int>>& edges) {
```

```
}
};
```

Java:

```
class Solution {
public int collectTheCoins(int[] coins, int[][] edges) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function collectTheCoins(coins: number[], edges: number[][]): number {

};
```

C#:

```
public class Solution {
public int CollectTheCoins(int[] coins, int[][] edges) {

}
}
```

C:

```
int collectTheCoins(int* coins, int coinsSize, int** edges, int edgesSize,
int* edgesColSize) {

}
```

Go:

```
func collectTheCoins(coins []int, edges [][]int) int {

}
```

Kotlin:

```
class Solution {
fun collectTheCoins(coins: IntArray, edges: Array<IntArray>): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
```

```
}
```

Ruby:

```
# @param {Integer[]} coins
# @param {Integer[][]} edges
# @return {Integer}
def collect_the_coins(coins, edges)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $coins
* @param Integer[][] $edges
* @return Integer
*/
function collectTheCoins($coins, $edges) {

}
}
```

Dart:

```
class Solution {
int collectTheCoins(List<int> coins, List<List<int>> edges) {

}
}
```

Scala:

```
object Solution {
def collectTheCoins(coins: Array[Int], edges: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec collect_the_coins(coins :: [integer], edges :: [[integer]]) :: integer
def collect_the_coins(coins, edges) do

end
end
```

Erlang:

```
-spec collect_the_coins(Coins :: [integer()], Edges :: [[integer()]]) ->
integer().
collect_the_coins(Coins, Edges) ->
.
```

Racket:

```
(define/contract (collect-the-coins coins edges)
(-> (listof exact-integer?) (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Collect Coins in a Tree
* Difficulty: Hard
* Tags: array, tree, graph, sort
*
* 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 collectTheCoins(vector<int>& coins, vector<vector<int>>& edges) {

}
};
```

Java Solution:

```
/**
* Problem: Collect Coins in a Tree
* Difficulty: Hard
* Tags: array, tree, graph, sort
*
* 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 collectTheCoins(int[] coins, int[][] edges) {

}
}
```

Python3 Solution:

```
"""
Problem: Collect Coins in a Tree
Difficulty: Hard
Tags: array, tree, graph, sort

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

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Collect Coins in a Tree
* Difficulty: Hard
* Tags: array, tree, graph, sort
*
* 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[]} coins
* @param {number[][]} edges
* @return {number}
*/
var collectTheCoins = function(coins, edges) {

};
```

TypeScript Solution:

```
/**
* Problem: Collect Coins in a Tree
* Difficulty: Hard
* Tags: array, tree, graph, sort
*
* 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 collectTheCoins(coins: number[], edges: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Collect Coins in a Tree
* Difficulty: Hard
* Tags: array, tree, graph, sort
*
* 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 CollectTheCoins(int[] coins, int[][] edges) {

}
}
```

C Solution:

```
/*
* Problem: Collect Coins in a Tree
* Difficulty: Hard
* Tags: array, tree, graph, sort
*
* 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 collectTheCoins(int* coins, int coinsSize, int** edges, int edgesSize,
int* edgesColSize) {

}
```

Go Solution:

```
// Problem: Collect Coins in a Tree
// Difficulty: Hard
// Tags: array, tree, graph, sort
//
// 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 collectTheCoins(coins []int, edges [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun collectTheCoins(coins: IntArray, edges: Array<IntArray>): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Collect Coins in a Tree
// Difficulty: Hard
// Tags: array, tree, graph, sort
//
// 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 collect_the_coins(coins: Vec<i32>, edges: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

```
# @return {Integer}
def collect_the_coins(coins, edges)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $coins
* @param Integer[][] $edges
* @return Integer
*/
function collectTheCoins($coins, $edges) {

}
}
```

Dart Solution:

```
class Solution {
int collectTheCoins(List<int> coins, List<List<int>> edges) {

}
}
```

Scala Solution:

```
object Solution {
def collectTheCoins(coins: Array[Int], edges: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec collect_the_coins(coins :: [integer], edges :: [[integer]]) :: integer
def collect_the_coins(coins, edges) do
```

```
end
end
```

Erlang Solution:

```
-spec collect_the_coins(Coins :: [integer()], Edges :: [[integer()]]) ->
integer().
collect_the_coins(Coins, Edges) ->
.
```

Racket Solution:

```
(define/contract (collect-the-coins coins edges)
(-> (listof exact-integer?) (listof (listof exact-integer?)) exact-integer?)
)
```