Problem 3241: Time Taken to Mark All Nodes

Problem Information

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

Problem Description

There exists an

undirected

tree with

n

nodes numbered

0

to

n - 1

. You are given a 2D integer array

edges

of length

n - 1

, where

edges[i] = [u_i, v_i] indicates that there is an edge between nodes u_i and v_i in the tree.

Initially, all nodes are unmarked. For each node i:

If

i

is odd, the node will get marked at time

x

if there is

at least

one node

adjacent

to it which was marked at time

$x - 1$

.

If

i

is even, the node will get marked at time

x

if there is

at least

one node

adjacent

to it which was marked at time

$x - 2$

.

Return an array

times

where

times[i]

is the time when all nodes get marked in the tree, if you mark node

i

at time

$t = 0$

.

Note

that the answer for each

times[i]

is

independent

, i.e. when you mark node

i

all other nodes are

unmarked

.

Example 1:

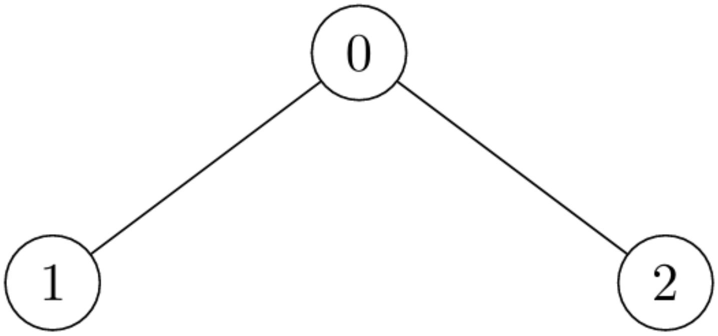
Input:

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

Output:

[2,4,3]

Explanation:



For

i = 0

:

Node 1 is marked at

t = 1

, and Node 2 at

t = 2

.

For

i = 1

:

Node 0 is marked at

t = 2

, and Node 2 at

t = 4

.

For

i = 2

:

Node 0 is marked at

t = 2

, and Node 1 at

t = 3

.

Example 2:

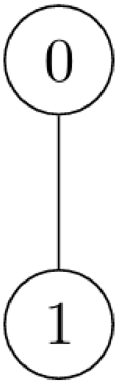
Input:

edges = [[0,1]]

Output:

[1,2]

Explanation:



For

i = 0

:

Node 1 is marked at

t = 1

.

For

i = 1

:

Node 0 is marked at

t = 2

.

Example 3:

Input:

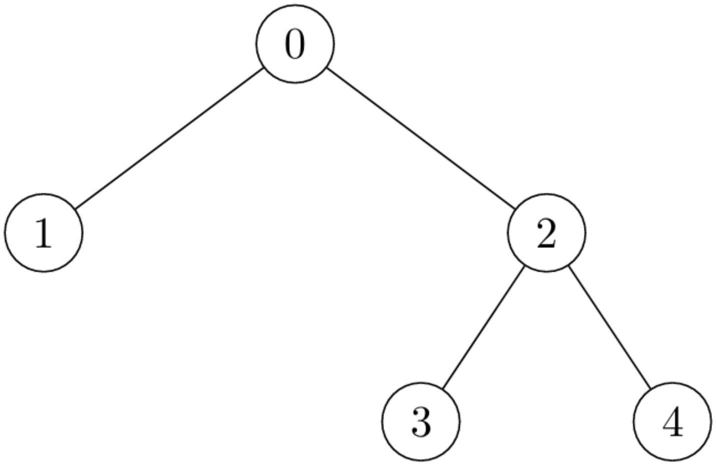
edges =

[[2,4],[0,1],[2,3],[0,2]]

Output:

[4,6,3,5,5]

Explanation:



Constraints:

2 <= n <= 10

5

edges.length == n - 1

edges[i].length == 2

0 <= edges[i][0], edges[i][1] <= n - 1

The input is generated such that

edges

represents a valid tree.

Code Snippets

C++:

```
class Solution {
public:
vector<int> timeTaken(vector<vector<int>>& edges) {

}
};
```

Java:

```
class Solution {
public int[] timeTaken(int[][] edges) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function timeTaken(edges: number[][]): number[] {

};
```

C#:

```
public class Solution {
public int[] TimeTaken(int[][] edges) {

}
}
```

C:

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

}
```

Go:

```
func timeTaken(edges [][]int) []int {

}
```

Kotlin:

```
class Solution {
fun timeTaken(edges: Array<IntArray>): IntArray {
}
}
```

Swift:

```
class Solution {
func timeTaken(_ edges: [[Int]]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn time_taken(edges: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[][]} edges
# @return {Integer[]}
def time_taken(edges)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $edges
* @return Integer[]
*/
function timeTaken($edges) {

}
}
```

Dart:

```
class Solution {
List<int> timeTaken(List<List<int>> edges) {

}
}
```

Scala:

```
object Solution {
def timeTaken(edges: Array[Array[Int]]): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec time_taken(edges :: [[integer]]) :: [integer]
def time_taken(edges) do

end
end
```

Erlang:

```
-spec time_taken(Edges :: [[integer()]]) -> [integer()].
time_taken(Edges) ->
.
```

Racket:

```
(define/contract (time-taken edges)
(-> (listof (listof exact-integer?)) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Time Taken to Mark All Nodes
* Difficulty: Hard
* Tags: array, tree, graph, dp, 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:
vector<int> timeTaken(vector<vector<int>>& edges) {

}
};
```

Java Solution:

```
/**
* Problem: Time Taken to Mark All Nodes
* Difficulty: Hard
* Tags: array, tree, graph, dp, 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[] timeTaken(int[][] edges) {

}
}
```

Python3 Solution:

```
"""
Problem: Time Taken to Mark All Nodes
Difficulty: Hard
Tags: array, tree, graph, dp, 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 timeTaken(self, edges: List[List[int]]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Time Taken to Mark All Nodes
* Difficulty: Hard
* Tags: array, tree, graph, dp, 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
* @return {number[]}
*/
var timeTaken = function(edges) {

};
```

TypeScript Solution:

```
/**
* Problem: Time Taken to Mark All Nodes
* Difficulty: Hard
* Tags: array, tree, graph, dp, 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 timeTaken(edges: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Time Taken to Mark All Nodes
* Difficulty: Hard
* Tags: array, tree, graph, dp, 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[] TimeTaken(int[][] edges) {

}
}
```

C Solution:

```
/*
* Problem: Time Taken to Mark All Nodes
* Difficulty: Hard
* Tags: array, tree, graph, dp, 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
```

```
*/

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

}
```

Go Solution:

```
// Problem: Time Taken to Mark All Nodes
// Difficulty: Hard
// Tags: array, tree, graph, dp, 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 timeTaken(edges [][]int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun timeTaken(edges: Array<IntArray>): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func timeTaken(_ edges: [[Int]]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Time Taken to Mark All Nodes
// Difficulty: Hard
// Tags: array, tree, graph, dp, 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 time_taken(edges: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} edges
# @return {Integer[]}
def time_taken(edges)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $edges
* @return Integer[]
*/
function timeTaken($edges) {

}
}
```

Dart Solution:

```
class Solution {
List<int> timeTaken(List<List<int>> edges) {

}
}
```

Scala Solution:

```
object Solution {
def timeTaken(edges: Array[Array[Int]]): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec time_taken(edges :: [[integer]]) :: [integer]
def time_taken(edges) do

end
end
```

Erlang Solution:

```
-spec time_taken(Edges :: [[integer()]]) -> [integer()].
time_taken(Edges) ->
.
```

Racket Solution:

```
(define/contract (time-taken edges)
(-> (listof (listof exact-integer?)) (listof exact-integer?))
)
```