Problem 3313: Find the Last Marked Nodes in Tree

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

. After every second, you mark all unmarked nodes which have

at least

one marked node

adjacent

to them.

Return an array

nodes

where

nodes[i]

is the last node to get marked in the tree, if you mark node

i

at time

t = 0

. If

nodes[i]

has

multiple

answers for any node

i

, you can choose

any

one answer.

Example 1:

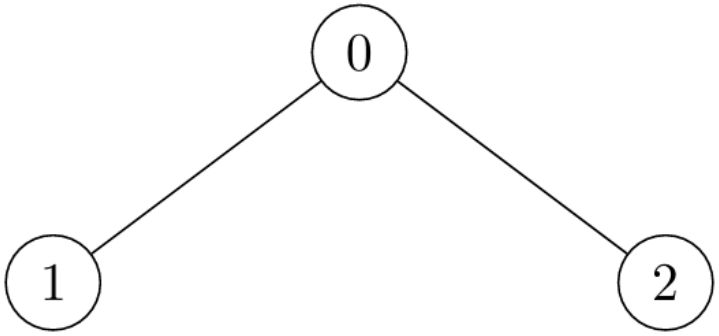
Input:

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

Output:

[2,2,1]

Explanation:



For

i = 0

, the nodes are marked in the sequence:

[0] -> [0,1,2]

. Either 1 or 2 can be the answer.

For

i = 1

, the nodes are marked in the sequence:

[1] -> [0,1] -> [0,1,2]

. Node 2 is marked last.

For

i = 2

, the nodes are marked in the sequence:

[2] -> [0,2] -> [0,1,2]

. Node 1 is marked last.

Example 2:

Input:

edges = [[0,1]]

Output:

[1,0]

Explanation:

For

i = 0

, the nodes are marked in the sequence:

[0] -> [0,1]

.

For

i = 1

, the nodes are marked in the sequence:

[1] -> [0,1]

.

Example 3:

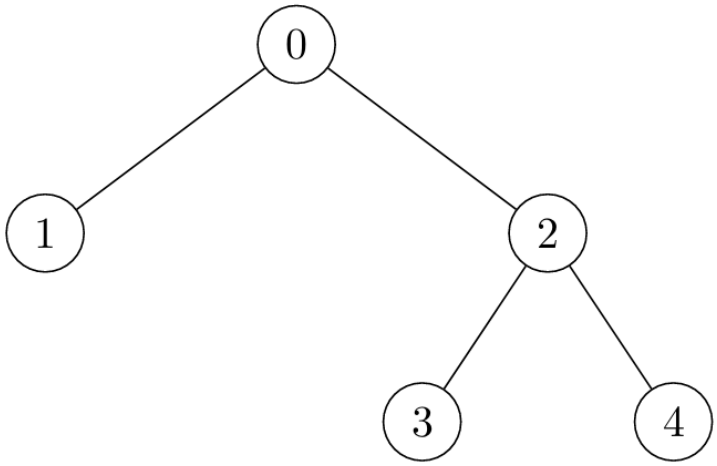
Input:

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

Output:

[3,3,1,1,1]

Explanation:



For

i = 0

, the nodes are marked in the sequence:

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

.

For

i = 1

, the nodes are marked in the sequence:

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

.

For

i = 2

, the nodes are marked in the sequence:

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

.

For

i = 3

, the nodes are marked in the sequence:

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

.

For

i = 4

, the nodes are marked in the sequence:

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

.

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> lastMarkedNodes(vector<vector<int>>& edges) {

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

```
var lastMarkedNodes = function(edges) {

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
```

```
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (last-marked-nodes edges)
(-> (listof (listof exact-integer?)) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find the Last Marked Nodes in Tree
* Difficulty: Hard
* Tags: array, tree, 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> lastMarkedNodes(vector<vector<int>>& edges) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Last Marked Nodes in Tree
* Difficulty: Hard
* Tags: array, tree, 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[] lastMarkedNodes(int[][] edges) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the Last Marked Nodes in Tree
Difficulty: Hard
Tags: array, tree, 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 lastMarkedNodes(self, edges: List[List[int]]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find the Last Marked Nodes in Tree
* Difficulty: Hard
* Tags: array, tree, 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[][]} edges
* @return {number[]}
*/
var lastMarkedNodes = function(edges) {

};
```

TypeScript Solution:

```
/**
* Problem: Find the Last Marked Nodes in Tree
* Difficulty: Hard
* Tags: array, tree, 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 lastMarkedNodes(edges: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Find the Last Marked Nodes in Tree
* Difficulty: Hard
* Tags: array, tree, 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[] LastMarkedNodes(int[][] edges) {

}
}
```

C Solution:

```
/*
* Problem: Find the Last Marked Nodes in Tree
* Difficulty: Hard
* Tags: array, tree, 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* lastMarkedNodes(int** edges, int edgesSize, int* edgesColSize, int*
returnSize) {

}
```

Go Solution:

```
// Problem: Find the Last Marked Nodes in Tree
// Difficulty: Hard
// Tags: array, tree, 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 lastMarkedNodes(edges [][]int) []int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find the Last Marked Nodes in Tree
// Difficulty: Hard
```

```
// Tags: array, tree, 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 last_marked_nodes(edges: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (last-marked-nodes edges)
(-> (listof (listof exact-integer?)) (listof exact-integer?))
)
```