Problem 310: Minimum Height Trees

Problem Information

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

Problem Description

A tree is an undirected graph in which any two vertices are connected by

exactly

one path. In other words, any connected graph without simple cycles is a tree.

Given a tree of

n

nodes labelled from

0

to

n - 1

, and an array of

n - 1

edges

where

edges[i] = [a

i

, b

i

]

indicates that there is an undirected edge between the two nodes

a

i

and

b

i

in the tree, you can choose any node of the tree as the root. When you select a node

x

as the root, the result tree has height

h

. Among all possible rooted trees, those with minimum height (i.e.

min(h)

) are called

minimum height trees

(MHTs).

Return

a list of all

MHTs'

root labels

. You can return the answer in

any order

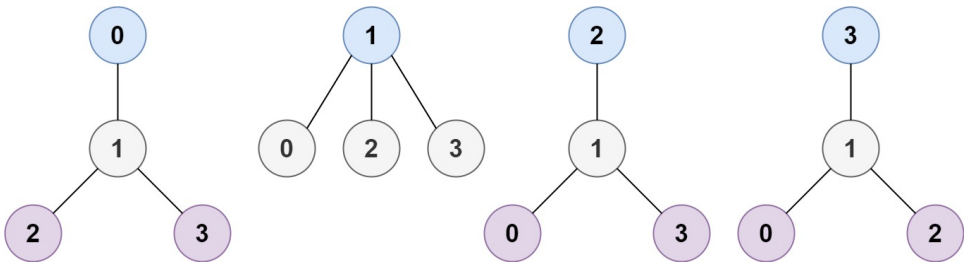
.

The

height

of a rooted tree is the number of edges on the longest downward path between the root and a leaf.

Example 1:



Input:

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

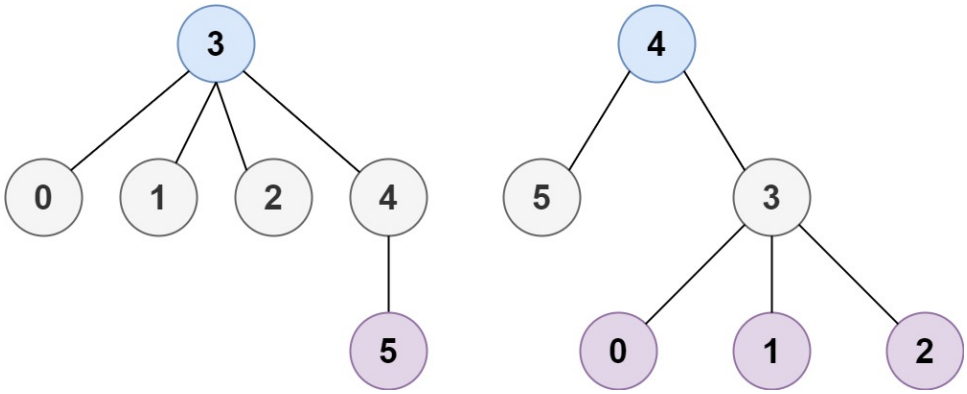
Output:

[1]

Explanation:

As shown, the height of the tree is 1 when the root is the node with label 1 which is the only MHT.

Example 2:



Input:

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

Output:

[3,4]

Constraints:

1 <= n <= 2 * 10

4

edges.length == n - 1

0 <= a

i

, b

i

< n

a

i

!= b

i

All the pairs

(a

i

, b

i

)

are distinct.

The given input is

guaranteed

to be a tree and there will be

no repeated

edges.

Code Snippets

C++:

```
class Solution {
public:
vector<int> findMinHeightTrees(int n, vector<vector<int>>& edges) {

}
};
```

Java:

```
class Solution {
public List<Integer> findMinHeightTrees(int n, int[][] edges) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function findMinHeightTrees(n: number, edges: number[][]): number[] {

};
```

C#:

```
public class Solution {
public IList<int> FindMinHeightTrees(int n, int[][] edges) {

}
}
```

C:

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

}
```

Go:

```
func findMinHeightTrees(n int, edges [][]int) []int {

}
```

Kotlin:

```
class Solution {
fun findMinHeightTrees(n: Int, edges: Array<IntArray>): List<Int> {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} edges
# @return {Integer[]}
def find_min_height_trees(n, edges)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edges
* @return Integer[]
*/
function findMinHeightTrees($n, $edges) {

}
}
```

Dart:

```
class Solution {
List<int> findMinHeightTrees(int n, List<List<int>> edges) {

}
}
```

Scala:

```
object Solution {
def findMinHeightTrees(n: Int, edges: Array[Array[Int]]): List[Int] = {
```

```
}
}
```

Elixir:

```
defmodule Solution do
@spec find_min_height_trees(n :: integer, edges :: [[integer]]) :: [integer]
def find_min_height_trees(n, edges) do

end
end
```

Erlang:

```
-spec find_min_height_trees(N :: integer(), Edges :: [[integer()]]) ->
[integer()].
find_min_height_trees(N, Edges) ->
.
```

Racket:

```
(define/contract (find-min-height-trees n edges)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Height Trees
* Difficulty: Medium
* Tags: array, tree, graph, sort, 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> findMinHeightTrees(int n, vector<vector<int>>& edges) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Height Trees
* Difficulty: Medium
* Tags: array, tree, graph, sort, 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 List<Integer> findMinHeightTrees(int n, int[][] edges) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Height Trees
Difficulty: Medium
Tags: array, tree, graph, sort, 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 findMinHeightTrees(self, n: int, edges: List[List[int]]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Height Trees
* Difficulty: Medium
* Tags: array, tree, graph, sort, 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} n
* @param {number[][]} edges
* @return {number[]}
*/
var findMinHeightTrees = function(n, edges) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Height Trees
* Difficulty: Medium
* Tags: array, tree, graph, sort, 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 findMinHeightTrees(n: number, edges: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Minimum Height Trees
* Difficulty: Medium
* Tags: array, tree, graph, sort, 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 IList<int> FindMinHeightTrees(int n, int[][] edges) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Height Trees
* Difficulty: Medium
* Tags: array, tree, graph, sort, 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* findMinHeightTrees(int n, int** edges, int edgesSize, int* edgesColSize,
int* returnSize) {
```

```
}
```

Go Solution:

```
// Problem: Minimum Height Trees
// Difficulty: Medium
// Tags: array, tree, graph, sort, 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 findMinHeightTrees(n int, edges [][]int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun findMinHeightTrees(n: Int, edges: Array<IntArray>): List<Int> {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Height Trees
// Difficulty: Medium
// Tags: array, tree, graph, sort, 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 find_min_height_trees(n: i32, edges: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} edges
# @return {Integer[]}
def find_min_height_trees(n, edges)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edges
* @return Integer[]
*/
function findMinHeightTrees($n, $edges) {

}
}
```

Dart Solution:

```
class Solution {
List<int> findMinHeightTrees(int n, List<List<int>> edges) {

}
}
```

Scala Solution:

```
object Solution {
def findMinHeightTrees(n: Int, edges: Array[Array[Int]]): List[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_min_height_trees(n :: integer, edges :: [[integer]]) :: [integer]
def find_min_height_trees(n, edges) do

end
end
```

Erlang Solution:

```
-spec find_min_height_trees(N :: integer(), Edges :: [[integer()]]) ->
[integer()].
find_min_height_trees(N, Edges) ->
.
```

Racket Solution:

```
(define/contract (find-min-height-trees n edges)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?))
)
```