Problem 1617: Count Subtrees With Max Distance Between Cities

Problem Information

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

Problem Description

There are

n

cities numbered from

1

to

n

. You are given an array

edges

of size

n-1

, where

edges[i] = [u

i

, v

i

]

represents a bidirectional edge between cities

u

i

and

v

i

. There exists a unique path between each pair of cities. In other words, the cities form a

tree

.

A

subtree

is a subset of cities where every city is reachable from every other city in the subset, where the path between each pair passes through only the cities from the subset. Two subtrees are different if there is a city in one subtree that is not present in the other.

For each

d

from

1

to

$n-1$

, find the number of subtrees in which the

maximum distance

between any two cities in the subtree is equal to

d

.

Return

an array of size

$n-1$

where the

d

th

element

(1-indexed)

is the number of subtrees in which the

maximum distance

between any two cities is equal to

d

.

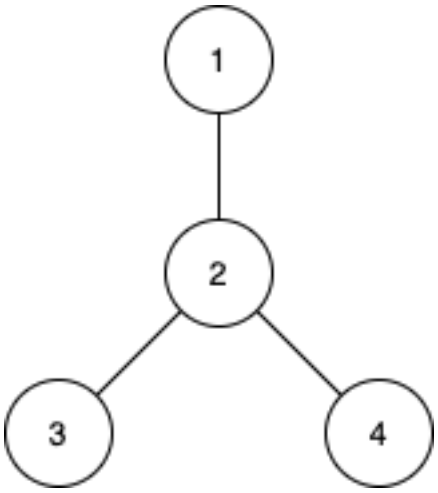
Notice

that the

distance

between the two cities is the number of edges in the path between them.

Example 1:



Input:

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

Output:

[3,4,0]

Explanation:

The subtrees with subsets {1,2}, {2,3} and {2,4} have a max distance of 1. The subtrees with subsets {1,2,3}, {1,2,4}, {2,3,4} and {1,2,3,4} have a max distance of 2. No subtree has two nodes where the max distance between them is 3.

Example 2:

Input:

n = 2, edges = [[1,2]]

Output:

[1]

Example 3:

Input:

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

Output:

[2,1]

Constraints:

2 <= n <= 15

edges.length == n-1

edges[i].length == 2

1 <= u

i

, v

i

<= n

All pairs

(u

i

, v

i

)

are distinct.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int[] countSubgraphsForEachDiameter(int n, int[][] edges) {

}
}
```

Python3:

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

Python:

```
class Solution(object):
def countSubgraphsForEachDiameter(self, n, edges):
```

```
"""
:type n: int
:type edges: List[List[int]]
:rtype: List[int]
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public int[] CountSubgraphsForEachDiameter(int n, int[][] edges) {

}
}
```

C:

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

```
}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Scala:

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

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Count Subtrees With Max Distance Between Cities
* Difficulty: Hard
* Tags: array, tree, dp
*
* 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> countSubgraphsForEachDiameter(int n, vector<vector<int>>& edges)
{
```

```
}
};
```

Java Solution:

```
/**
* Problem: Count Subtrees With Max Distance Between Cities
* Difficulty: Hard
* Tags: array, tree, dp
*
* 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[] countSubgraphsForEachDiameter(int n, int[][] edges) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Subtrees With Max Distance Between Cities
Difficulty: Hard
Tags: array, tree, dp

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Subtrees With Max Distance Between Cities
* Difficulty: Hard
* Tags: array, tree, dp
*
* 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} n
* @param {number[][]} edges
* @return {number[]}
*/
var countSubgraphsForEachDiameter = function(n, edges) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Subtrees With Max Distance Between Cities
* Difficulty: Hard
* Tags: array, tree, dp
*
* 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 countSubgraphsForEachDiameter(n: number, edges: number[][]):
```

```
number[] {

};
```

C# Solution:

```
/*
* Problem: Count Subtrees With Max Distance Between Cities
* Difficulty: Hard
* Tags: array, tree, dp
*
* 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[] CountSubgraphsForEachDiameter(int n, int[][] edges) {

}
}
```

C Solution:

```
/*
* Problem: Count Subtrees With Max Distance Between Cities
* Difficulty: Hard
* Tags: array, tree, dp
*
* 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* countSubgraphsForEachDiameter(int n, int** edges, int edgesSize, int*
edgesColSize, int* returnSize){
```

```
}
```

Go Solution:

```
// Problem: Count Subtrees With Max Distance Between Cities
// Difficulty: Hard
// Tags: array, tree, dp
//
// 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 countSubgraphsForEachDiameter(n int, edges [][]int) []int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Count Subtrees With Max Distance Between Cities
// Difficulty: Hard
// Tags: array, tree, dp
//
// 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 count_subgraphs_for_each_diameter(n: i32, edges: Vec<Vec<i32>>) ->
Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Scala Solution:

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

}
}
```