Problem 3067: Count Pairs of Connectable Servers in a Weighted Tree Network

Problem Information

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

Problem Description

You are given an unrooted weighted tree with

n

vertices representing servers numbered from

0

to

n - 1

, an array

edges

where

edges[i] = [a

i

, b

i

, weight

i

]

represents a bidirectional edge between vertices

a

i

and

b

i

of weight

weight

i

. You are also given an integer

signalSpeed

.

Two servers

a

and

b

are

connectable

through a server

c

if:

a < b

,

a != c

and

b != c

.

The distance from

c

to

a

is divisible by

signalSpeed

.

The distance from

c

to

b

is divisible by

signalSpeed

.

The path from

c

to

b

and the path from

c

to

a

do not share any edges.

Return

an integer array

count

of length

n

where

count[i]

is the

number

of server pairs that are

connectable

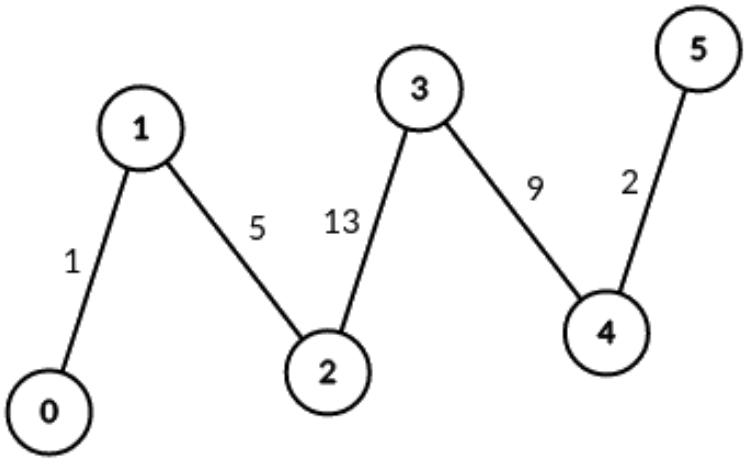
through

the server

i

.

Example 1:



Input:

edges = [[0,1,1],[1,2,5],[2,3,13],[3,4,9],[4,5,2]], signalSpeed = 1

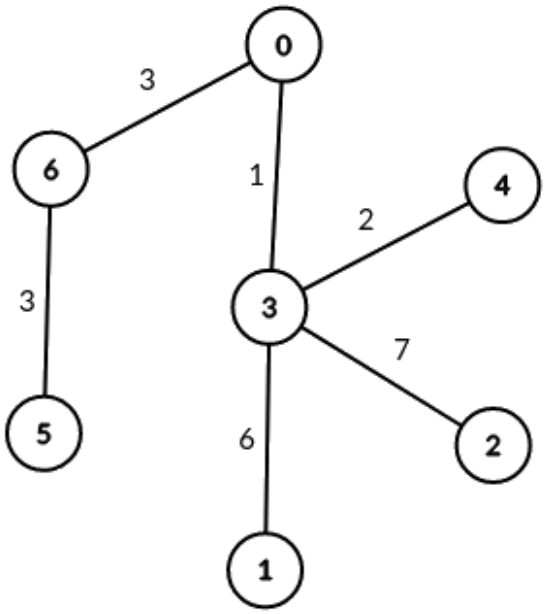
Output:

[0,4,6,6,4,0]

Explanation:

Since signalSpeed is 1, count[c] is equal to the number of pairs of paths that start at c and do not share any edges. In the case of the given path graph, count[c] is equal to the number of servers to the left of c multiplied by the servers to the right of c.

Example 2:



Input:

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

Output:

[2,0,0,0,0,0,2]

Explanation:

Through server 0, there are 2 pairs of connectable servers: (4, 5) and (4, 6). Through server 6, there are 2 pairs of connectable servers: (4, 5) and (0, 5). It can be shown that no two servers are connectable through servers other than 0 and 6.

Constraints:

2 <= n <= 1000

edges.length == n - 1

edges[i].length == 3

0 <= a

i

, b

i

< n

edges[i] = [a

i

, b

i

, weight

i

]

1 <= weight

i

<= 10

6

1 <= signalSpeed <= 10

6

The input is generated such that

edges

represents a valid tree.

Code Snippets

C++:

```
class Solution {
public:
vector<int> countPairsOfConnectableServers(vector<vector<int>>& edges, int
signalSpeed) {

}
};
```

Java:

```
class Solution {
public int[] countPairsOfConnectableServers(int[][] edges, int signalSpeed) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function countPairsOfConnectableServers(edges: number[][], signalSpeed:
number): number[] {

};
```

C#:

```
public class Solution {
public int[] CountPairsOfConnectableServers(int[][] edges, int signalSpeed) {

}
}
```

C:

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

}
```

Go:

```
func countPairsOfConnectableServers(edges [][]int, signalSpeed int) []int {

}
```

Kotlin:

```
class Solution {
fun countPairsOfConnectableServers(edges: Array<IntArray>, signalSpeed: Int):
IntArray {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[][]} edges
# @param {Integer} signal_speed
# @return {Integer[]}
def count_pairs_of_connectable_servers(edges, signal_speed)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $edges
* @param Integer $signalSpeed
* @return Integer[]
*/
function countPairsOfConnectableServers($edges, $signalSpeed) {

}
}
```

Dart:

```
class Solution {
List<int> countPairsOfConnectableServers(List<List<int>> edges, int
signalSpeed) {

}
}
```

Scala:

```
object Solution {
def countPairsOfConnectableServers(edges: Array[Array[Int]], signalSpeed:
Int): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_pairs_of_connectable_servers(edges :: [[integer]], signal_speed
```

```
:: integer) :: [integer]
def count_pairs_of_connectable_servers(edges, signal_speed) do

end
end
```

Erlang:

```
-spec count_pairs_of_connectable_servers(Edges :: [[integer()]], SignalSpeed
:: integer()) -> [integer()].
count_pairs_of_connectable_servers(Edges, SignalSpeed) ->
.
```

Racket:

```
(define/contract (count-pairs-of-connectable-servers edges signalSpeed)
(-> (listof (listof exact-integer?)) exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Pairs of Connectable Servers in a Weighted Tree Network
* Difficulty: Medium
* Tags: array, tree, graph, 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> countPairsOfConnectableServers(vector<vector<int>>& edges, int
signalSpeed) {

}
};
```

Java Solution:

```
/**
* Problem: Count Pairs of Connectable Servers in a Weighted Tree Network
* Difficulty: Medium
* Tags: array, tree, graph, 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[] countPairsOfConnectableServers(int[][] edges, int signalSpeed) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Pairs of Connectable Servers in a Weighted Tree Network
Difficulty: Medium
Tags: array, tree, graph, 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 countPairsOfConnectableServers(self, edges: List[List[int]], signalSpeed:
int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Count Pairs of Connectable Servers in a Weighted Tree Network
* Difficulty: Medium
* Tags: array, tree, graph, 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
* @param {number} signalSpeed
* @return {number[]}
*/
var countPairsOfConnectableServers = function(edges, signalSpeed) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Pairs of Connectable Servers in a Weighted Tree Network
* Difficulty: Medium
* Tags: array, tree, graph, 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 countPairsOfConnectableServers(edges: number[][], signalSpeed:
number): number[] {

};
```

C# Solution:

```
/*
* Problem: Count Pairs of Connectable Servers in a Weighted Tree Network
* Difficulty: Medium
* Tags: array, tree, graph, 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[] CountPairsOfConnectableServers(int[][] edges, int signalSpeed) {

}
}
```

C Solution:

```
/*
* Problem: Count Pairs of Connectable Servers in a Weighted Tree Network
* Difficulty: Medium
* Tags: array, tree, graph, 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* countPairsOfConnectableServers(int** edges, int edgesSize, int*
edgesColSize, int signalSpeed, int* returnSize) {

}
```

Go Solution:

```
// Problem: Count Pairs of Connectable Servers in a Weighted Tree Network
// Difficulty: Medium
// Tags: array, tree, graph, 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 countPairsOfConnectableServers(edges [][]int, signalSpeed int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun countPairsOfConnectableServers(edges: Array<IntArray>, signalSpeed: Int):
IntArray {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Count Pairs of Connectable Servers in a Weighted Tree Network
// Difficulty: Medium
// Tags: array, tree, graph, 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 count_pairs_of_connectable_servers(edges: Vec<Vec<i32>>, signal_speed:
i32) -> Vec<i32> {
```

```
}
}
```

Ruby Solution:

```
# @param {Integer[][]} edges
# @param {Integer} signal_speed
# @return {Integer[]}
def count_pairs_of_connectable_servers(edges, signal_speed)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $edges
* @param Integer $signalSpeed
* @return Integer[]
*/
function countPairsOfConnectableServers($edges, $signalSpeed) {

}
}
```

Dart Solution:

```
class Solution {
List<int> countPairsOfConnectableServers(List<List<int>> edges, int
signalSpeed) {

}
}
```

Scala Solution:

```
object Solution {
def countPairsOfConnectableServers(edges: Array[Array[Int]], signalSpeed:
Int): Array[Int] = {
```

```
}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_pairs_of_connectable_servers(edges :: [[integer]], signal_speed
:: integer) :: [integer]
def count_pairs_of_connectable_servers(edges, signal_speed) do

end
end
```

Erlang Solution:

```
-spec count_pairs_of_connectable_servers(Edges :: [[integer()]], SignalSpeed
:: integer()) -> [integer()].
count_pairs_of_connectable_servers(Edges, SignalSpeed) ->
.
```

Racket Solution:

```
(define/contract (count-pairs-of-connectable-servers edges signalSpeed)
(-> (listof (listof exact-integer?)) exact-integer? (listof exact-integer?))
)
```