Problem 3373: Maximize the Number of Target Nodes After Connecting Trees II

Problem Information

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

Problem Description

There exist two

undirected

trees with

n

and

m

nodes, labeled from

[0, n - 1]

and

[0, m - 1]

, respectively.

You are given two 2D integer arrays

edges1

and

edges2

of lengths

n - 1

and

m - 1

, respectively, where

edges1[i] = [a

i

, b

i

]

indicates that there is an edge between nodes

a

i

and

b

i

in the first tree and

edges2[i] = [u

i

, v

i

]

indicates that there is an edge between nodes

u

i

and

v

i

in the second tree.

Node

u

is

target

to node

v

if the number of edges on the path from

u

to

v

is even.

Note

that a node is

always

target

to itself.

Return an array of

n

integers

answer

, where

answer[i]

is the

maximum

possible number of nodes that are

target

to node

i

of the first tree if you had to connect one node from the first tree to another node in the second tree.

Note

that queries are independent from each other. That is, for every query you will remove the added edge before proceeding to the next query.

Example 1:

Input:

edges1 = [[0,1],[0,2],[2,3],[2,4]], edges2 = [[0,1],[0,2],[0,3],[2,7],[1,4],[4,5],[4,6]]

Output:

[8,7,7,8,8]

Explanation:

For

i = 0

, connect node 0 from the first tree to node 0 from the second tree.

For

i = 1

, connect node 1 from the first tree to node 4 from the second tree.

For

i = 2

, connect node 2 from the first tree to node 7 from the second tree.

For

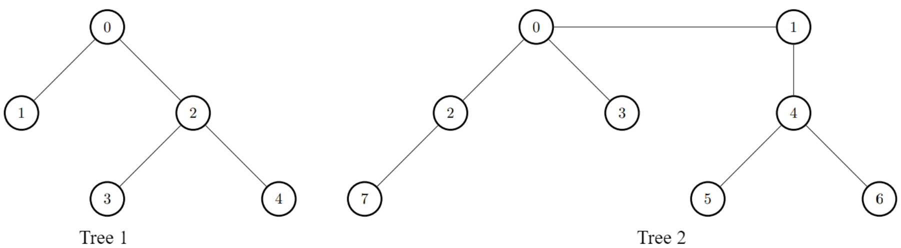
i = 3

, connect node 3 from the first tree to node 0 from the second tree.

For

i = 4

, connect node 4 from the first tree to node 4 from the second tree.



Example 2:

Input:

edges1 = [[0,1],[0,2],[0,3],[0,4]], edges2 = [[0,1],[1,2],[2,3]]

Output:

[3,6,6,6,6]

Explanation:

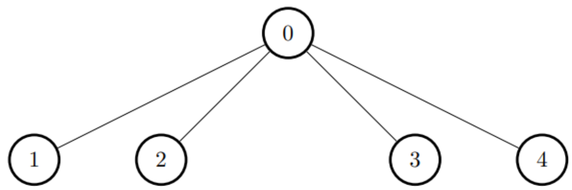
For every

i

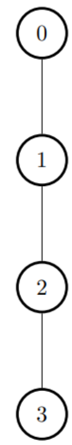
, connect node

i

of the first tree with any node of the second tree.



Tree 1



Tree 2

Constraints:

2 <= n, m <= 10

5

edges1.length == n - 1

edges2.length == m - 1

edges1[i].length == edges2[i].length == 2

edges1[i] = [a

i

, b

i

]

0 <= a

i

, b

i

< n

edges2[i] = [u

i

, v

i

]

0 <= u

i

, v

i

< m

The input is generated such that

edges1

and

edges2

represent valid trees.

Code Snippets

C++:

```
class Solution {
public:
vector<int> maxTargetNodes(vector<vector<int>>& edges1, vector<vector<int>>&
edges2) {

}
};
```

Java:

```
class Solution {
public int[] maxTargetNodes(int[][] edges1, int[][] edges2) {

}
}
```

Python3:

```
class Solution:
def maxTargetNodes(self, edges1: List[List[int]], edges2: List[List[int]]) ->
List[int]:
```

Python:

```
class Solution(object):
def maxTargetNodes(self, edges1, edges2):
"""
:type edges1: List[List[int]]
:type edges2: List[List[int]]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[][]} edges1
* @param {number[][]} edges2
* @return {number[]}
*/
```

```
var maxTargetNodes = function(edges1, edges2) {

};
```

TypeScript:

```
function maxTargetNodes(edges1: number[][], edges2: number[][]): number[] {

};
```

C#:

```
public class Solution {
public int[] MaxTargetNodes(int[][] edges1, int[][] edges2) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* maxTargetNodes(int** edges1, int edges1Size, int* edges1ColSize, int**
edges2, int edges2Size, int* edges2ColSize, int* returnSize) {

}
```

Go:

```
func maxTargetNodes(edges1 [][]int, edges2 [][]int) []int {

}
```

Kotlin:

```
class Solution {
fun maxTargetNodes(edges1: Array<IntArray>, edges2: Array<IntArray>):
IntArray {

}
}
```

Swift:

```
class Solution {
func maxTargetNodes(_ edges1: [[Int]], _ edges2: [[Int]]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn max_target_nodes(edges1: Vec<Vec<i32>>, edges2: Vec<Vec<i32>>) ->
Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[][]} edges1
# @param {Integer[][]} edges2
# @return {Integer[]}
def max_target_nodes(edges1, edges2)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $edges1
* @param Integer[][] $edges2
* @return Integer[]
*/
function maxTargetNodes($edges1, $edges2) {

}
}
```

Dart:

```
class Solution {
List<int> maxTargetNodes(List<List<int>> edges1, List<List<int>> edges2) {

}
}
```

Scala:

```
object Solution {
def maxTargetNodes(edges1: Array[Array[Int]], edges2: Array[Array[Int]]):
Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_target_nodes(edges1 :: [[integer]], edges2 :: [[integer]]) ::
[integer]
def max_target_nodes(edges1, edges2) do

end
end
```

Erlang:

```
-spec max_target_nodes(Edges1 :: [[integer()]], Edges2 :: [[integer()]]) ->
[integer()].
max_target_nodes(Edges1, Edges2) ->
.
```

Racket:

```
(define/contract (max-target-nodes edges1 edges2)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)) (listof
exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximize the Number of Target Nodes After Connecting Trees II
* 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> maxTargetNodes(vector<vector<int>>& edges1, vector<vector<int>>&
edges2) {

}
};
```

Java Solution:

```
/**
* Problem: Maximize the Number of Target Nodes After Connecting Trees II
* 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[] maxTargetNodes(int[][] edges1, int[][] edges2) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximize the Number of Target Nodes After Connecting Trees II
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 maxTargetNodes(self, edges1: List[List[int]], edges2: List[List[int]]) ->
List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxTargetNodes(self, edges1, edges2):
"""
:type edges1: List[List[int]]
:type edges2: List[List[int]]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Maximize the Number of Target Nodes After Connecting Trees II
* 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[][]} edges1
* @param {number[][]} edges2
* @return {number[]}
*/
var maxTargetNodes = function(edges1, edges2) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Maximize the Number of Target Nodes After Connecting Trees II
* 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 maxTargetNodes(edges1: number[][], edges2: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Maximize the Number of Target Nodes After Connecting Trees II
* 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[] MaxTargetNodes(int[][] edges1, int[][] edges2) {

}
}
```

C Solution:

```
/*
* Problem: Maximize the Number of Target Nodes After Connecting Trees II
```

```
* 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* maxTargetNodes(int** edges1, int edges1Size, int* edges1ColSize, int**
edges2, int edges2Size, int* edges2ColSize, int* returnSize) {

}
```

Go Solution:

```go
// Problem: Maximize the Number of Target Nodes After Connecting Trees II
// 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 maxTargetNodes(edges1 [][]int, edges2 [][]int) []int {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun maxTargetNodes(edges1: Array<IntArray>, edges2: Array<IntArray>):
IntArray {

}
}
```

Swift Solution:

```
class Solution {
func maxTargetNodes(_ edges1: [[Int]], _ edges2: [[Int]]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Maximize the Number of Target Nodes After Connecting Trees II
// 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 max_target_nodes(edges1: Vec<Vec<i32>>, edges2: Vec<Vec<i32>>) ->
Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} edges1
# @param {Integer[][]} edges2
# @return {Integer[]}
def max_target_nodes(edges1, edges2)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $edges1
* @param Integer[][] $edges2
* @return Integer[]
*/
function maxTargetNodes($edges1, $edges2) {
```

```
}
}
```

Dart Solution:

```
class Solution {
List<int> maxTargetNodes(List<List<int>> edges1, List<List<int>> edges2) {

}
}
```

Scala Solution:

```
object Solution {
def maxTargetNodes(edges1: Array[Array[Int]], edges2: Array[Array[Int]]):
Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_target_nodes(edges1 :: [[integer]], edges2 :: [[integer]]) ::
[integer]
def max_target_nodes(edges1, edges2) do

end
end
```

Erlang Solution:

```
-spec max_target_nodes(Edges1 :: [[integer()]], Edges2 :: [[integer()]]) ->
[integer()].
max_target_nodes(Edges1, Edges2) ->
.
```

Racket Solution:

```
(define/contract (max-target-nodes edges1 edges2)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)) (listof
exact-integer?))
)
```