Problem 3559: Number of Ways to Assign Edge Weights II

Problem Information

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

Problem Description

There is an undirected tree with

n

nodes labeled from 1 to

n

, rooted at node 1. The tree is represented by 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

.

Initially, all edges have a weight of 0. You must assign each edge a weight of either

1

or

2

.

The

cost

of a path between any two nodes

u

and

v

is the total weight of all edges in the path connecting them.

You are given a 2D integer array

queries

. For each

queries[i] = [u

i

, v

i

]

, determine the number of ways to assign weights to edges

in the path

such that the cost of the path between

u

i

and

v

i

is

odd

.

Return an array

answer

, where

answer[i]

is the number of valid assignments for

queries[i]

.

Since the answer may be large, apply

modulo

10

9

+ 7

to each

answer[i]

.

Note:

For each query, disregard all edges

not

in the path between node

u

i

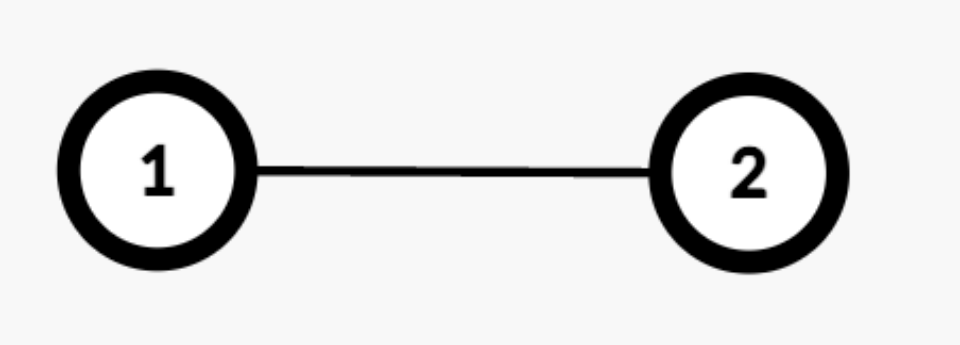
and

v

i

.

Example 1:



Input:

edges = [[1,2]], queries = [[1,1],[1,2]]

Output:

[0,1]

Explanation:

Query

[1,1]

: The path from Node 1 to itself consists of no edges, so the cost is 0. Thus, the number of valid assignments is 0.

Query

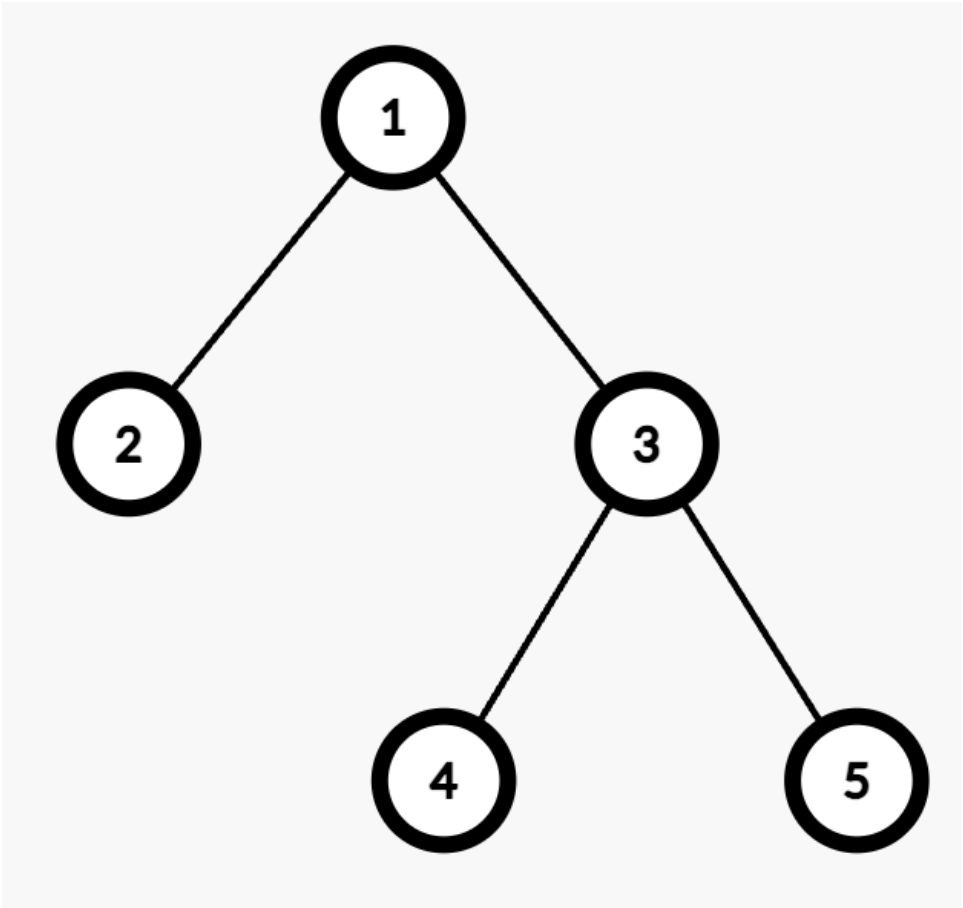
[1,2]

: The path from Node 1 to Node 2 consists of one edge (

$1 \rightarrow 2$

). Assigning weight 1 makes the cost odd, while 2 makes it even. Thus, the number of valid assignments is 1.

Example 2:



Input:

edges = [[1,2],[1,3],[3,4],[3,5]], queries = [[1,4],[3,4],[2,5]]

Output:

[2,1,4]

Explanation:

Query

[1,4]

: The path from Node 1 to Node 4 consists of two edges (

$1 \rightarrow 3$

and

$3 \rightarrow 4$

). Assigning weights (1,2) or (2,1) results in an odd cost. Thus, the number of valid assignments is 2.

Query

[3,4]

: The path from Node 3 to Node 4 consists of one edge (

$3 \rightarrow 4$

). Assigning weight 1 makes the cost odd, while 2 makes it even. Thus, the number of valid assignments is 1.

Query

[2,5]

: The path from Node 2 to Node 5 consists of three edges (

$2 \rightarrow 1$, $1 \rightarrow 3$

, and

$3 \rightarrow 5$

). Assigning (1,2,2), (2,1,2), (2,2,1), or (1,1,1) makes the cost odd. Thus, the number of valid assignments is 4.

Constraints:

2 <= n <= 10

5

edges.length == n - 1

edges[i] == [u

i

, v

i

]

1 <= queries.length <= 10

5

queries[i] == [u

i

, v

i

]

1 <= u

i

, v

i

<= n

edges

represents a valid tree.

Code Snippets

C++:

```
class Solution {
public:
vector<int> assignEdgeWeights(vector<vector<int>>& edges,
vector<vector<int>>& queries) {

}
};
```

Java:

```
class Solution {
public int[] assignEdgeWeights(int[][] edges, int[][] queries) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function assignEdgeWeights(edges: number[][], queries: number[][]): number[]
{

};
```

C#:

```
public class Solution {
public int[] AssignEdgeWeights(int[][] edges, int[][] queries) {

}
}
```

C:

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

}
```

Go:

```
func assignEdgeWeights(edges [][]int, queries [][]int) []int {

}
```

Kotlin:

```
class Solution {
fun assignEdgeWeights(edges: Array<IntArray>, queries: Array<IntArray>):
IntArray {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[][]} edges
# @param {Integer[][]} queries
# @return {Integer[]}
def assign_edge_weights(edges, queries)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $edges
* @param Integer[][] $queries
* @return Integer[]
*/
function assignEdgeWeights($edges, $queries) {

}
}
```

Dart:

```
class Solution {
List<int> assignEdgeWeights(List<List<int>> edges, List<List<int>> queries) {

}
}
```

Scala:

```
object Solution {
def assignEdgeWeights(edges: Array[Array[Int]], queries: Array[Array[Int]]):
Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec assign_edge_weights(edges :: [[integer]], queries :: [[integer]]) ::
[integer]
def assign_edge_weights(edges, queries) do

end
end
```

Erlang:

```
-spec assign_edge_weights(Edges :: [[integer()]], Queries :: [[integer()]])
-> [integer()].
```

```
assign_edge_weights(Edges, Queries) ->
.
```

Racket:

```
(define/contract (assign-edge-weights edges queries)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)) (listof
exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Ways to Assign Edge Weights II
* Difficulty: Hard
* Tags: array, tree, dp, math, search
*
* 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> assignEdgeWeights(vector<vector<int>>& edges,
vector<vector<int>>& queries) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Ways to Assign Edge Weights II
* Difficulty: Hard
* Tags: array, tree, dp, math, search
*
* 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[] assignEdgeWeights(int[][] edges, int[][] queries) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Ways to Assign Edge Weights II
Difficulty: Hard
Tags: array, tree, dp, math, search

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of Ways to Assign Edge Weights II
* Difficulty: Hard
* Tags: array, tree, dp, math, search
*
* 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[][]} edges
* @param {number[][]} queries
* @return {number[]}
*/
var assignEdgeWeights = function(edges, queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Ways to Assign Edge Weights II
* Difficulty: Hard
* Tags: array, tree, dp, math, search
*
* 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 assignEdgeWeights(edges: number[][], queries: number[][]): number[]
{

};
```

C# Solution:

```
/*
* Problem: Number of Ways to Assign Edge Weights II
* Difficulty: Hard
* Tags: array, tree, dp, math, search
```

```
*
* 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[] AssignEdgeWeights(int[][] edges, int[][] queries) {

}
}
```

C Solution:

```
/*
* Problem: Number of Ways to Assign Edge Weights II
* Difficulty: Hard
* Tags: array, tree, dp, math, search
*
* 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* assignEdgeWeights(int** edges, int edgesSize, int* edgesColSize, int**
queries, int queriesSize, int* queriesColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Number of Ways to Assign Edge Weights II
// Difficulty: Hard
// Tags: array, tree, dp, math, search
//
// 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 assignEdgeWeights(edges [][]int, queries [][]int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun assignEdgeWeights(edges: Array<IntArray>, queries: Array<IntArray>):
IntArray {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Number of Ways to Assign Edge Weights II
// Difficulty: Hard
// Tags: array, tree, dp, math, search
//
// 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 assign_edge_weights(edges: Vec<Vec<i32>>, queries: Vec<Vec<i32>>) ->
Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} edges
# @param {Integer[][]} queries
# @return {Integer[]}
def assign_edge_weights(edges, queries)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $edges
* @param Integer[][] $queries
* @return Integer[]
*/
function assignEdgeWeights($edges, $queries) {

}
}
```

Dart Solution:

```
class Solution {
List<int> assignEdgeWeights(List<List<int>> edges, List<List<int>> queries) {

}
}
```

Scala Solution:

```
object Solution {
def assignEdgeWeights(edges: Array[Array[Int]], queries: Array[Array[Int]]):
Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec assign_edge_weights(edges :: [[integer]], queries :: [[integer]]) ::
```

```
[integer]
def assign_edge_weights(edges, queries) do

end
end
```

Erlang Solution:

```
-spec assign_edge_weights(Edges :: [[integer()]], Queries :: [[integer()]])
-> [integer()].
assign_edge_weights(Edges, Queries) ->
.
```

Racket Solution:

```
(define/contract (assign-edge-weights edges queries)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)) (listof
exact-integer?))
)
```