Problem 2858: Minimum Edge Reversals So Every Node Is Reachable

Problem Information

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

Problem Description

There is a

simple directed graph

with

n

nodes labeled from

0

to

n - 1

. The graph would form a

tree

if its edges were bi-directional.

You are given an integer

n

and a

2D

integer array

edges

, where

edges[i] = [u

i

, v

i

]

represents a

directed edge

going from node

u

i

to node

v

i

.

An

edge reversal

changes the direction of an edge, i.e., a directed edge going from node

u_i

to node

v_i

becomes a directed edge going from node

v_i

to node

u_i

.

For every node

i

in the range

$[0, n - 1]$

, your task is to

independently

calculate the

minimum

number of

edge reversals

required so it is possible to reach any other node starting from node

i

through a

sequence

of

directed edges

.

Return

an integer array

answer

, where

answer[i]

is the

minimum

number of

edge reversals

required so it is possible to reach any other node starting from node

i

through a

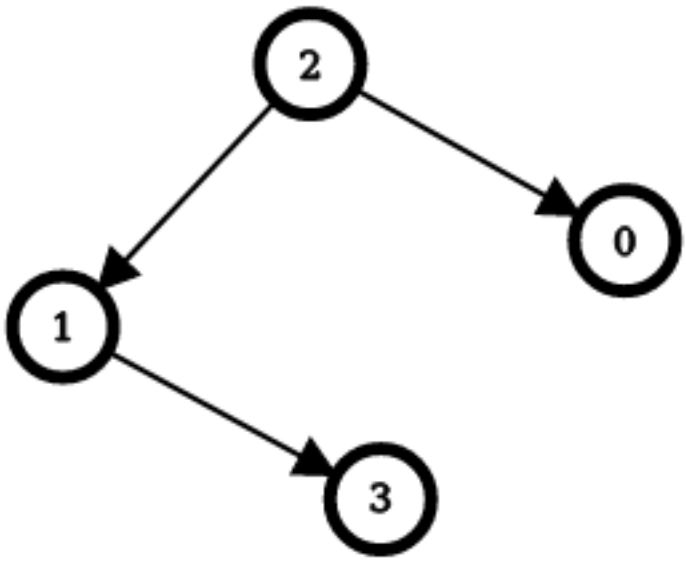
sequence

of

directed edges

.

Example 1:



Input:

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

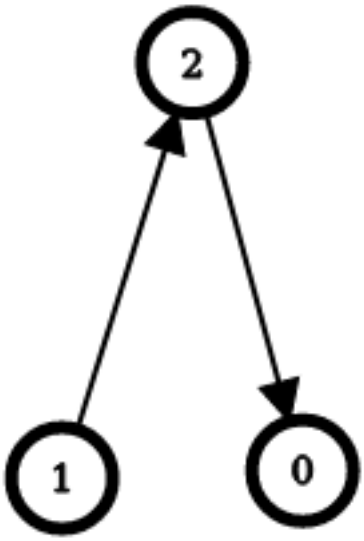
Output:

[1,1,0,2]

Explanation:

The image above shows the graph formed by the edges. For node 0: after reversing the edge [2,0], it is possible to reach any other node starting from node 0. So, answer[0] = 1. For node 1: after reversing the edge [2,1], it is possible to reach any other node starting from node 1. So, answer[1] = 1. For node 2: it is already possible to reach any other node starting from node 2. So, answer[2] = 0. For node 3: after reversing the edges [1,3] and [2,1], it is possible to reach any other node starting from node 3. So, answer[3] = 2.

Example 2:



Input:

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

Output:

[2,0,1]

Explanation:

The image above shows the graph formed by the edges. For node 0: after reversing the edges [2,0] and [1,2], it is possible to reach any other node starting from node 0. So, answer[0] = 2. For node 1: it is already possible to reach any other node starting from node 1. So, answer[1] = 0. For node 2: after reversing the edge [1, 2], it is possible to reach any other node starting from node 2. So, answer[2] = 1.

Constraints:

2 <= n <= 10

5

edges.length == n - 1

edges[i].length == 2

0 <= u

i

== edges[i][0] < n

0 <= v

i

== edges[i][1] < n

u

i

!= v

i

The input is generated such that if the edges were bi-directional, the graph would be a tree.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

```
};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
```

```
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (min-edge-reversals n edges)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Edge Reversals So Every Node Is Reachable
* Difficulty: Hard
* Tags: array, tree, graph, dp, 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> minEdgeReversals(int n, vector<vector<int>>& edges) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Edge Reversals So Every Node Is Reachable
* Difficulty: Hard
* Tags: array, tree, graph, dp, 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[] minEdgeReversals(int n, int[][] edges) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Edge Reversals So Every Node Is Reachable
Difficulty: Hard
Tags: array, tree, graph, dp, 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 minEdgeReversals(self, n: int, edges: List[List[int]]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Edge Reversals So Every Node Is Reachable
* Difficulty: Hard
* Tags: array, tree, graph, dp, 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} n
* @param {number[][]} edges
* @return {number[]}
*/
var minEdgeReversals = function(n, edges) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Edge Reversals So Every Node Is Reachable
* Difficulty: Hard
* Tags: array, tree, graph, dp, 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 minEdgeReversals(n: number, edges: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Minimum Edge Reversals So Every Node Is Reachable
* Difficulty: Hard
* Tags: array, tree, graph, dp, 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[] MinEdgeReversals(int n, int[][] edges) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Edge Reversals So Every Node Is Reachable
* Difficulty: Hard
* Tags: array, tree, graph, dp, 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* minEdgeReversals(int n, int** edges, int edgesSize, int* edgesColSize,
int* returnSize) {

}
```

Go Solution:

```
// Problem: Minimum Edge Reversals So Every Node Is Reachable
// Difficulty: Hard
// Tags: array, tree, graph, dp, 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 minEdgeReversals(n int, edges [][]int) []int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Edge Reversals So Every Node Is Reachable
// Difficulty: Hard
// Tags: array, tree, graph, dp, 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 min_edge_reversals(n: i32, edges: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

```
}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (min-edge-reversals n edges)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?))
)
```