Problem 1466: Reorder Routes to Make All Paths Lead to the City Zero

Problem Information

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

Problem Description

There are

n

cities numbered from

0

to

n - 1

and

n - 1

roads such that there is only one way to travel between two different cities (this network form a tree). Last year, The ministry of transport decided to orient the roads in one direction because they are too narrow.

Roads are represented by

connections

where

connections[i] = [a

i

, b

i

]

represents a road from city

a

i

to city

b

i

.

This year, there will be a big event in the capital (city

0

), and many people want to travel to this city.

Your task consists of reorienting some roads such that each city can visit the city

0

. Return the

minimum

number of edges changed.

It's

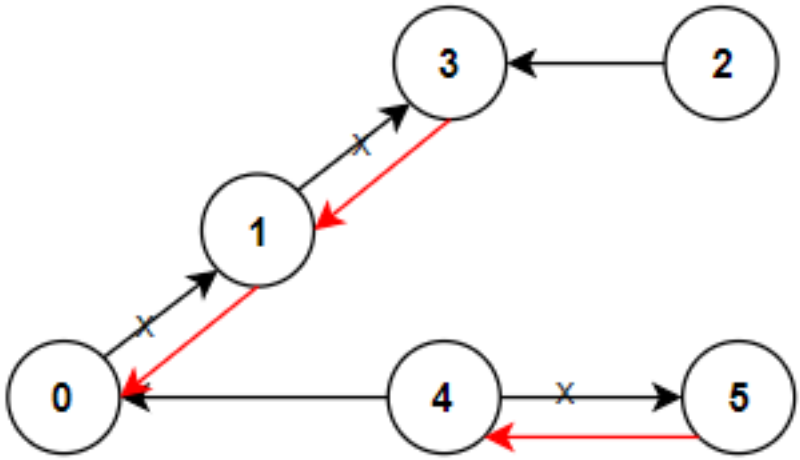
guaranteed

that each city can reach city

0

after reorder.

Example 1:



Input:

n = 6, connections = [[0,1],[1,3],[2,3],[4,0],[4,5]]

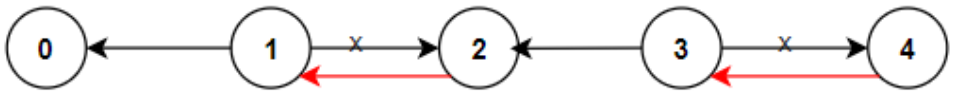
Output:

3

Explanation:

Change the direction of edges show in red such that each node can reach the node 0 (capital).

Example 2:



Input:

n = 5, connections = [[1,0],[1,2],[3,2],[3,4]]

Output:

2

Explanation:

Change the direction of edges show in red such that each node can reach the node 0 (capital).

Example 3:

Input:

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

Output:

0

Constraints:

2 <= n <= 5 * 10

4

connections.length == n - 1

connections[i].length == 2

0 <= a

i

, b

i

<= n - 1

a

i

!= b

i

Code Snippets

C++:

```
class Solution {
public:
int minReorder(int n, vector<vector<int>>& connections) {

}
};
```

Java:

```
class Solution {
public int minReorder(int n, int[][] connections) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minReorder(n: number, connections: number[][]): number {

};
```

C#:

```
public class Solution {
public int MinReorder(int n, int[][] connections) {

}
}
```

C:

```
int minReorder(int n, int** connections, int connectionsSize, int*
connectionsColSize) {
```

```
}
```

Go:

```
func minReorder(n int, connections [][]int) int {

}
```

Kotlin:

```
class Solution {
fun minReorder(n: Int, connections: Array<IntArray>): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} connections
# @return {Integer}
def min_reorder(n, connections)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $connections
* @return Integer
*/
function minReorder($n, $connections) {

}
}
```

Dart:

```
class Solution {
int minReorder(int n, List<List<int>> connections) {

}
}
```

Scala:

```
object Solution {
def minReorder(n: Int, connections: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_reorder(n :: integer, connections :: [[integer]]) :: integer
def min_reorder(n, connections) do

end
end
```

Erlang:

```
-spec min_reorder(N :: integer(), Connections :: [[integer()]]) -> integer().
min_reorder(N, Connections) ->
.
```

Racket:

```
(define/contract (min-reorder n connections)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Reorder Routes to Make All Paths Lead to the City Zero
* Difficulty: Medium
* Tags: tree, graph, search
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
int minReorder(int n, vector<vector<int>>& connections) {

}
};
```

Java Solution:

```
/**
* Problem: Reorder Routes to Make All Paths Lead to the City Zero
* Difficulty: Medium
* Tags: tree, graph, search
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public int minReorder(int n, int[][] connections) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Reorder Routes to Make All Paths Lead to the City Zero
Difficulty: Medium
Tags: tree, graph, search

Approach: DFS or BFS traversal
Time Complexity: O(n) where n is number of nodes
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
def minReorder(self, n: int, connections: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Reorder Routes to Make All Paths Lead to the City Zero
* Difficulty: Medium
* Tags: tree, graph, search
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
```

```
*/

/**
* @param {number} n
* @param {number[][]} connections
* @return {number}
*/
var minReorder = function(n, connections) {

};
```

TypeScript Solution:

```
/**
* Problem: Reorder Routes to Make All Paths Lead to the City Zero
* Difficulty: Medium
* Tags: tree, graph, search
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

function minReorder(n: number, connections: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Reorder Routes to Make All Paths Lead to the City Zero
* Difficulty: Medium
* Tags: tree, graph, search
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public int MinReorder(int n, int[][] connections) {
```

```
}
}
```

C Solution:

```
/*
* Problem: Reorder Routes to Make All Paths Lead to the City Zero
* Difficulty: Medium
* Tags: tree, graph, search
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

int minReorder(int n, int** connections, int connectionsSize, int*
connectionsColSize) {

}
```

Go Solution:

```
// Problem: Reorder Routes to Make All Paths Lead to the City Zero
// Difficulty: Medium
// Tags: tree, graph, search
//
// Approach: DFS or BFS traversal
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(h) for recursion stack where h is height

func minReorder(n int, connections [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minReorder(n: Int, connections: Array<IntArray>): Int {

}
```

```
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Reorder Routes to Make All Paths Lead to the City Zero
// Difficulty: Medium
// Tags: tree, graph, search
//
// Approach: DFS or BFS traversal
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn min_reorder(n: i32, connections: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} connections
# @return {Integer}
def min_reorder(n, connections)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
```

```
* @param Integer[][] $connections
* @return Integer
*/
function minReorder($n, $connections) {

}
}
```

Dart Solution:

```
class Solution {
int minReorder(int n, List<List<int>> connections) {

}
}
```

Scala Solution:

```
object Solution {
def minReorder(n: Int, connections: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_reorder(n :: integer, connections :: [[integer]]) :: integer
def min_reorder(n, connections) do

end
end
```

Erlang Solution:

```
-spec min_reorder(N :: integer(), Connections :: [[integer()]]) -> integer().
min_reorder(N, Connections) ->
.
```

Racket Solution:

```
(define/contract (min-reorder n connections)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```