Problem 1579: Remove Max Number of Edges to Keep Graph Fully Traversable

Problem Information

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

Problem Description

Alice and Bob have an undirected graph of

n

nodes and three types of edges:

Type 1: Can be traversed by Alice only.

Type 2: Can be traversed by Bob only.

Type 3: Can be traversed by both Alice and Bob.

Given an array

edges

where

edges[i] = [type

i

, u

i

, v

i

]

represents a bidirectional edge of type

type

i

between nodes

u

i

and

v

i

, find the maximum number of edges you can remove so that after removing the edges, the graph can still be fully traversed by both Alice and Bob. The graph is fully traversed by Alice and Bob if starting from any node, they can reach all other nodes.

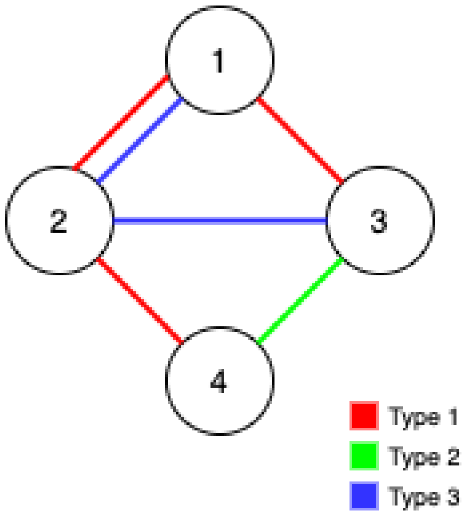
Return

the maximum number of edges you can remove, or return

-1

if Alice and Bob cannot fully traverse the graph.

Example 1:



Input:

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

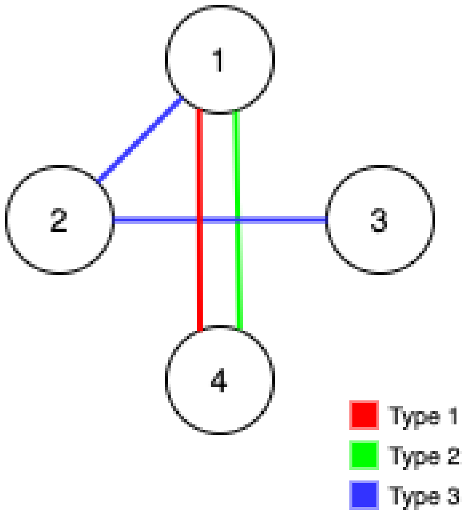
Output:

2

Explanation:

If we remove the 2 edges [1,1,2] and [1,1,3]. The graph will still be fully traversable by Alice and Bob. Removing any additional edge will not make it so. So the maximum number of edges we can remove is 2.

Example 2:



Input:

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

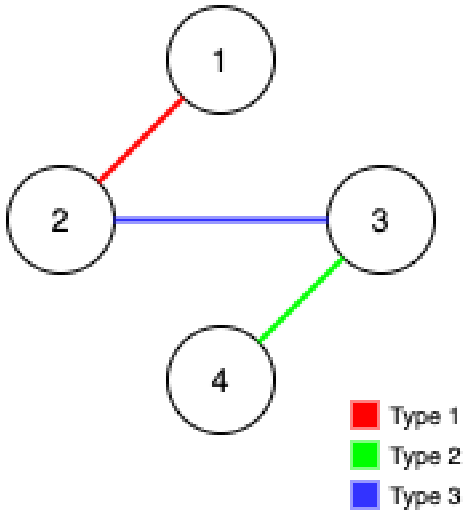
Output:

0

Explanation:

Notice that removing any edge will not make the graph fully traversable by Alice and Bob.

Example 3:



Input:

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

Output:

-1

Explanation:

In the current graph, Alice cannot reach node 4 from the other nodes. Likewise, Bob cannot reach 1. Therefore it's impossible to make the graph fully traversable.

Constraints:

1 <= n <= 10

5

1 <= edges.length <= min(10

5

, 3 * n * (n - 1) / 2)

edges[i].length == 3

1 <= type

i

<= 3

1 <= u

i

< v

i

<= n

All tuples

(type

i

, u

i

, v

i

)

are distinct.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

```
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int maxNumEdgesToRemove(int n, int** edges, int edgesSize, int* edgesColSize)
{

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (max-num-edges-to-remove n edges)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Remove Max Number of Edges to Keep Graph Fully Traversable
* Difficulty: Hard
* Tags: array, graph
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Remove Max Number of Edges to Keep Graph Fully Traversable
* Difficulty: Hard
* Tags: array, graph
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Remove Max Number of Edges to Keep Graph Fully Traversable
```

```
Difficulty: Hard
Tags: array, graph

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Remove Max Number of Edges to Keep Graph Fully Traversable
* Difficulty: Hard
* Tags: array, graph
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

TypeScript Solution:

```
/**
* Problem: Remove Max Number of Edges to Keep Graph Fully Traversable
* Difficulty: Hard
* Tags: array, graph
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function maxNumEdgesToRemove(n: number, edges: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Remove Max Number of Edges to Keep Graph Fully Traversable
* Difficulty: Hard
* Tags: array, graph
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
}
```

C Solution:

```
/*
* Problem: Remove Max Number of Edges to Keep Graph Fully Traversable
* Difficulty: Hard
* Tags: array, graph
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int maxNumEdgesToRemove(int n, int** edges, int edgesSize, int* edgesColSize)
{

}
```

Go Solution:

```
// Problem: Remove Max Number of Edges to Keep Graph Fully Traversable
// Difficulty: Hard
// Tags: array, graph
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func maxNumEdgesToRemove(n int, edges [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}

}
```

Rust Solution:

```
// Problem: Remove Max Number of Edges to Keep Graph Fully Traversable
// Difficulty: Hard
// Tags: array, graph
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (max-num-edges-to-remove n edges)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```