Problem 1761: Minimum Degree of a Connected Trio in a Graph

Problem Information

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

Problem Description

You are given an undirected graph. You are given an integer

n

which is the number of nodes in the graph and an array

edges

, where each

edges[i] = [u

i

, v

i

]

indicates that there is an undirected edge between

u

i

and

v

i

.

A

connected trio

is a set of

three

nodes where there is an edge between

every

pair of them.

The

degree of a connected trio

is the number of edges where one endpoint is in the trio, and the other is not.

Return

the

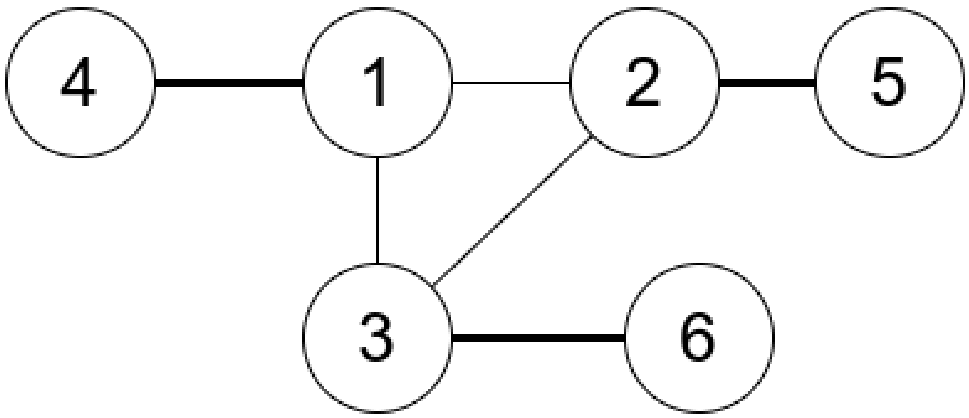
minimum

degree of a connected trio in the graph, or

-1

if the graph has no connected trios.

Example 1:



Input:

n = 6, edges = [[1,2],[1,3],[3,2],[4,1],[5,2],[3,6]]

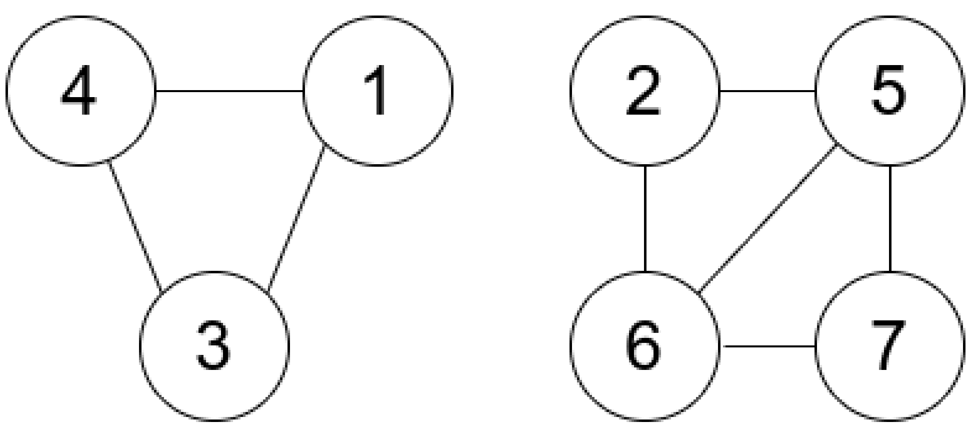
Output:

3

Explanation:

There is exactly one trio, which is [1,2,3]. The edges that form its degree are bolded in the figure above.

Example 2:



Input:

n = 7, edges = [[1,3],[4,1],[4,3],[2,5],[5,6],[6,7],[7,5],[2,6]]

Output:

0

Explanation:

There are exactly three trios: 1) [1,4,3] with degree 0. 2) [2,5,6] with degree 2. 3) [5,6,7] with degree 2.

Constraints:

2 <= n <= 400

edges[i].length == 2

1 <= edges.length <= n * (n-1) / 2

1 <= u

i

, v

i

<= n

u

i

!= v

i

There are no repeated edges.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} edges
* @return {number}
*/
```

```
var minTrioDegree = function(n, edges) {

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

```



}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (min-trio-degree n edges)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Degree of a Connected Trio in a Graph
* Difficulty: Hard
* Tags: array, graph, dp
*
* 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 minTrioDegree(int n, vector<vector<int>>& edges) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Degree of a Connected Trio in a Graph
* Difficulty: Hard
* Tags: array, graph, dp
*
* 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 minTrioDegree(int n, int[][] edges) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Degree of a Connected Trio in a Graph
Difficulty: Hard
Tags: array, graph, dp

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 minTrioDegree(self, n: int, edges: List[List[int]]) -> int:
```

```
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Degree of a Connected Trio in a Graph
* Difficulty: Hard
* Tags: array, graph, dp
*
* 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 minTrioDegree = function(n, edges) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Degree of a Connected Trio in a Graph
* Difficulty: Hard
* Tags: array, graph, dp
*
```

```
* 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 minTrioDegree(n: number, edges: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Degree of a Connected Trio in a Graph
* Difficulty: Hard
* Tags: array, graph, dp
*
* 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 MinTrioDegree(int n, int[][] edges) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Degree of a Connected Trio in a Graph
* Difficulty: Hard
* Tags: array, graph, dp
*
* 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
*/

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

```
}
```

Go Solution:

```
// Problem: Minimum Degree of a Connected Trio in a Graph
// Difficulty: Hard
// Tags: array, graph, dp
//
// 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 minTrioDegree(n int, edges [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Degree of a Connected Trio in a Graph
// Difficulty: Hard
// Tags: array, graph, dp
//
// 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_trio_degree(n: i32, edges: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (min-trio-degree n edges)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```