Problem 1319: Number of Operations to Make Network Connected

Problem Information

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

Problem Description

There are

n

computers numbered from

0

to

n - 1

connected by ethernet cables

connections

forming a network where

connections[i] = [a

i

, b

i

]

represents a connection between computers

a

i

and

b

i

. Any computer can reach any other computer directly or indirectly through the network.

You are given an initial computer network

connections

. You can extract certain cables between two directly connected computers, and place them between any pair of disconnected computers to make them directly connected.

Return

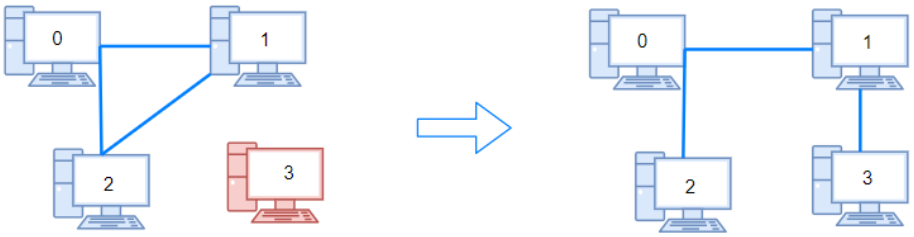
the minimum number of times you need to do this in order to make all the computers connected

. If it is not possible, return

-1

.

Example 1:



Input:

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

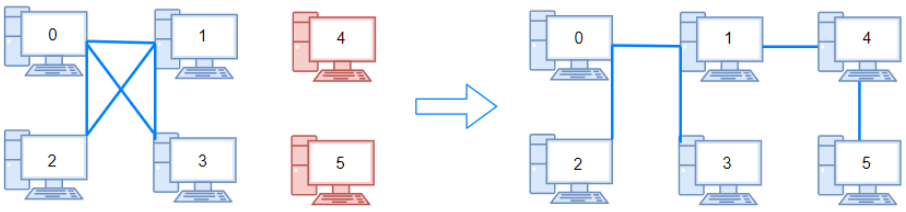
Output:

1

Explanation:

Remove cable between computer 1 and 2 and place between computers 1 and 3.

Example 2:



Input:

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

Output:

2

Example 3:

Input:

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

Output:

-1

Explanation:

There are not enough cables.

Constraints:

1 <= n <= 10

5

1 <= connections.length <= min(n * (n - 1) / 2, 10

5

)

connections[i].length == 2

0 <= a

i

, b

i

< n

a

i

!= b

i

There are no repeated connections.

No two computers are connected by more than one cable.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

```
}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (make-connected n connections)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Operations to Make Network Connected
* Difficulty: Medium
* Tags: graph, search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Number of Operations to Make Network Connected
* Difficulty: Medium
* Tags: graph, search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Number of Operations to Make Network Connected
Difficulty: Medium
Tags: graph, search
```

```
Approach: Optimized algorithm based on problem constraints
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of Operations to Make Network Connected
* Difficulty: Medium
* Tags: graph, search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Number of Operations to Make Network Connected
* Difficulty: Medium
* Tags: graph, search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

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

};
```

C# Solution:

```
/*
* Problem: Number of Operations to Make Network Connected
* Difficulty: Medium
* Tags: graph, search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
}
```

C Solution:

```
/*
* Problem: Number of Operations to Make Network Connected
* Difficulty: Medium
* Tags: graph, search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
```

```
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
```

Go Solution:

```
// Problem: Number of Operations to Make Network Connected
// Difficulty: Medium
// Tags: graph, search
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

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

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Number of Operations to Make Network Connected
// Difficulty: Medium
// Tags: graph, search
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

```
}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (make-connected n connections)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```