Problem 2493: Divide Nodes Into the Maximum Number of Groups

Problem Information

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

Problem Description

You are given a positive integer

n

representing the number of nodes in an

undirected

graph. The nodes are labeled from

1

to

n

.

You are also given a 2D integer array

edges

, where

edges[i] = [a

i,

b

i

]

indicates that there is a

bidirectional

edge between nodes

a

i

and

b

i

.

Notice

that the given graph may be disconnected.

Divide the nodes of the graph into

m

groups (

1-indexed

) such that:

Each node in the graph belongs to exactly one group.

For every pair of nodes in the graph that are connected by an edge

[a

i,

b

i

]

, if

a

i

belongs to the group with index

x

, and

b

i

belongs to the group with index

y

, then

|y - x| = 1

.

Return

the maximum number of groups (i.e., maximum

m

) into which you can divide the nodes

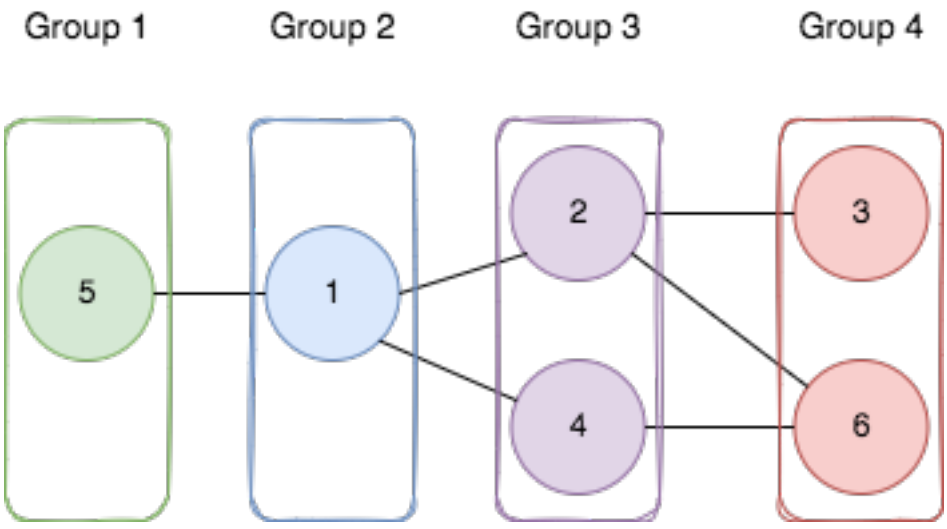
. Return

-1

if it is impossible to group the nodes with the given conditions

.

Example 1:



Input:

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

Output:

4

Explanation:

As shown in the image we: - Add node 5 to the first group. - Add node 1 to the second group. - Add nodes 2 and 4 to the third group. - Add nodes 3 and 6 to the fourth group. We can see that every edge is satisfied. It can be shown that that if we create a fifth group and move any node from the third or fourth group to it, at least on of the edges will not be satisfied.

Example 2:

Input:

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

Output:

-1

Explanation:

If we add node 1 to the first group, node 2 to the second group, and node 3 to the third group to satisfy the first two edges, we can see that the third edge will not be satisfied. It can be shown that no grouping is possible.

Constraints:

1 <= n <= 500

1 <= edges.length <= 10

4

edges[i].length == 2

1 <= a

i

, b

i

<= n

a

i

!= b

i

There is at most one edge between any pair of vertices.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (magnificent-sets n edges)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Divide Nodes Into the Maximum Number of Groups
* Difficulty: Hard
* Tags: array, graph, search
*
* 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 magnificentSets(int n, vector<vector<int>>& edges) {

}
};
```

Java Solution:

```
/**
* Problem: Divide Nodes Into the Maximum Number of Groups
* Difficulty: Hard
* Tags: array, graph, search
*
* 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 magnificentSets(int n, int[][] edges) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Divide Nodes Into the Maximum Number of Groups
Difficulty: Hard
Tags: array, graph, search

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Divide Nodes Into the Maximum Number of Groups
* Difficulty: Hard
* Tags: array, graph, search
*
* 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 magnificentSets = function(n, edges) {

};
```

TypeScript Solution:

```
/**
* Problem: Divide Nodes Into the Maximum Number of Groups
* Difficulty: Hard
* Tags: array, graph, search
*
* 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 magnificentSets(n: number, edges: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Divide Nodes Into the Maximum Number of Groups
* Difficulty: Hard
* Tags: array, graph, search
*
* 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 MagnificentSets(int n, int[][] edges) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Divide Nodes Into the Maximum Number of Groups
* Difficulty: Hard
* Tags: array, graph, search
*
* 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 magnificentSets(int n, int** edges, int edgesSize, int* edgesColSize) {

}
```

Go Solution:

```
// Problem: Divide Nodes Into the Maximum Number of Groups
// Difficulty: Hard
// Tags: array, graph, search
//
// 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 magnificentSets(n int, edges [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Divide Nodes Into the Maximum Number of Groups
// Difficulty: Hard
// Tags: array, graph, search
//
// 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 magnificent_sets(n: i32, edges: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

```
}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (magnificent-sets n edges)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```