Problem 2508: Add Edges to Make Degrees of All Nodes Even

Problem Information

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

Problem Description

There is an

undirected

graph consisting of

n

nodes numbered from

1

to

n

. You are given the integer

n

and a

2D

array

edges

where

edges[i] = [a

i

, b

i

]

indicates that there is an edge between nodes

a

i

and

b

i

. The graph can be disconnected.

You can add

at most

two additional edges (possibly none) to this graph so that there are no repeated edges and no self-loops.

Return

true

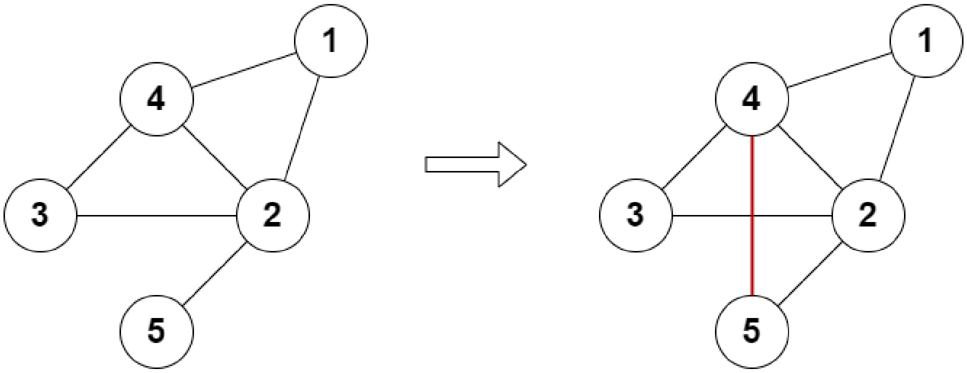
if it is possible to make the degree of each node in the graph even, otherwise return

false

.

The degree of a node is the number of edges connected to it.

Example 1:



Input:

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

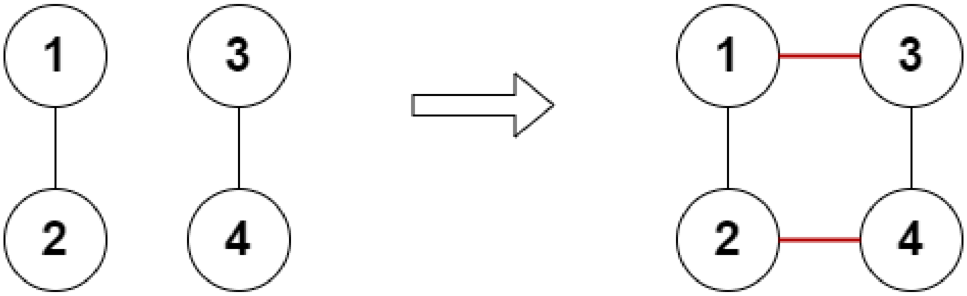
Output:

true

Explanation:

The above diagram shows a valid way of adding an edge. Every node in the resulting graph is connected to an even number of edges.

Example 2:



Input:

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

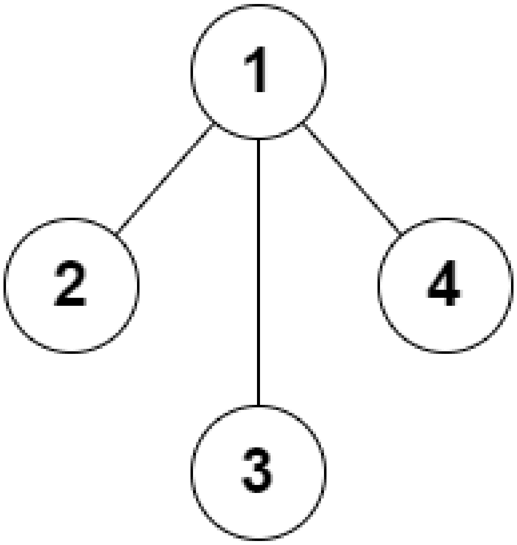
Output:

true

Explanation:

The above diagram shows a valid way of adding two edges.

Example 3:



Input:

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

Output:

false

Explanation:

It is not possible to obtain a valid graph with adding at most 2 edges.

Constraints:

3 <= n <= 10

5

2 <= edges.length <= 10

5

edges[i].length == 2

1 <= a

i

, b

i

<= n

a

i

!= b

i

There are no repeated edges.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public boolean isPossible(int n, List<List<Integer>> edges) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public bool IsPossible(int n, IList<IList<int>> edges) {

}
}
```

C:

```
bool isPossible(int n, int** edges, int edgesSize, int* edgesColSize) {

}
```

Go:

```
func isPossible(n int, edges [][]int) bool {

}
```

Kotlin:

```
class Solution {
fun isPossible(n: Int, edges: List<List<Int>>): Boolean {

}
}
```

Swift:

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

}

}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

```
object Solution {
def isPossible(n: Int, edges: List[List[Int]]): Boolean = {

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (is-possible n edges)
(-> exact-integer? (listof (listof exact-integer?)) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Add Edges to Make Degrees of All Nodes Even
* Difficulty: Hard
* Tags: array, graph, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/
```

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

}
};
```

Java Solution:

```
/**
* Problem: Add Edges to Make Degrees of All Nodes Even
* Difficulty: Hard
* Tags: array, graph, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public boolean isPossible(int n, List<List<Integer>> edges) {

}
}
```

Python3 Solution:

```
"""
Problem: Add Edges to Make Degrees of All Nodes Even
Difficulty: Hard
Tags: array, graph, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Add Edges to Make Degrees of All Nodes Even
* Difficulty: Hard
* Tags: array, graph, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Add Edges to Make Degrees of All Nodes Even
* Difficulty: Hard
* Tags: array, graph, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/
```

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

};
```

C# Solution:

```
/*
* Problem: Add Edges to Make Degrees of All Nodes Even
* Difficulty: Hard
* Tags: array, graph, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public bool IsPossible(int n, IList<IList<int>> edges) {

}
}
```

C Solution:

```
/*
* Problem: Add Edges to Make Degrees of All Nodes Even
* Difficulty: Hard
* Tags: array, graph, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

bool isPossible(int n, int** edges, int edgesSize, int* edgesColSize) {

}
```

Go Solution:

```
// Problem: Add Edges to Make Degrees of All Nodes Even
// Difficulty: Hard
// Tags: array, graph, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func isPossible(n int, edges [][]int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun isPossible(n: Int, edges: List<List<Int>>): Boolean {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Add Edges to Make Degrees of All Nodes Even
// Difficulty: Hard
// Tags: array, graph, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

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

}

}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

```
object Solution {
def isPossible(n: Int, edges: List[List[Int]]): Boolean = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (is-possible n edges)
(-> exact-integer? (listof (listof exact-integer?)) boolean?)
)
```