Problem 886: Possible Bipartition

Problem Information

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

Problem Description

We want to split a group of

n

people (labeled from

1

to

n

) into two groups of

any size

. Each person may dislike some other people, and they should not go into the same group.

Given the integer

n

and the array

dislikes

where

dislikes[i] = [a

i

, b

i

]

indicates that the person labeled

a

i

does not like the person labeled

b

i

, return

true

if it is possible to split everyone into two groups in this way

.

Example 1:

Input:

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

Output:

true

Explanation:

The first group has [1,4], and the second group has [2,3].

Example 2:

Input:

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

Output:

false

Explanation:

We need at least 3 groups to divide them. We cannot put them in two groups.

Constraints:

1 <= n <= 2000

0 <= dislikes.length <= 10

4

dislikes[i].length == 2

1 <= a

i

< b

i

<= n

All the pairs of

dislikes

are

unique

.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public boolean possibleBipartition(int n, int[][] dislikes) {

}
}
```

Python3:

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

Python:

```
class Solution(object):
def possibleBipartition(self, n, dislikes):
```

```
"""
:type n: int
:type dislikes: List[List[int]]
:rtype: bool
"""
```

JavaScript:

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

};
```

TypeScript:

```
function possibleBipartition(n: number, dislikes: number[][]): boolean {

};
```

C#:

```
public class Solution {
public bool PossibleBipartition(int n, int[][] dislikes) {

}
}
```

C:

```
bool possibleBipartition(int n, int** dislikes, int dislikesSize, int*
dislikesColSize) {

}
```

Go:

```
func possibleBipartition(n int, dislikes [][]int) bool {
}
```

Kotlin:

```
class Solution {
fun possibleBipartition(n: Int, dislikes: Array<IntArray>): Boolean {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
bool possibleBipartition(int n, List<List<int>> dislikes) {

}
}
```

Scala:

```
object Solution {
def possibleBipartition(n: Int, dislikes: Array[Array[Int]]): Boolean = {

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec possible_bipartition(N :: integer(), Dislikes :: [[integer()]]) ->
boolean().
possible_bipartition(N, Dislikes) ->
.
```

Racket:

```
(define/contract (possible-bipartition n dislikes)
(-> exact-integer? (listof (listof exact-integer?)) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Possible Bipartition
* Difficulty: Medium
* 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:
bool possibleBipartition(int n, vector<vector<int>>& dislikes) {

}
};
```

Java Solution:

```
/**
* Problem: Possible Bipartition
* Difficulty: Medium
* 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 boolean possibleBipartition(int n, int[][] dislikes) {

}
}
```

Python3 Solution:

```
"""
Problem: Possible Bipartition
Difficulty: Medium
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 possibleBipartition(self, n: int, dislikes: List[List[int]]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Possible Bipartition
* Difficulty: Medium
* 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[][]} dislikes
* @return {boolean}
*/
var possibleBipartition = function(n, dislikes) {

};
```

TypeScript Solution:

```
/**
* Problem: Possible Bipartition
* Difficulty: Medium
* 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 possibleBipartition(n: number, dislikes: number[][]): boolean {

};
```

C# Solution:

```
/*
* Problem: Possible Bipartition
* Difficulty: Medium
* 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 bool PossibleBipartition(int n, int[][] dislikes) {

}

}
```

C Solution:

```
/*
* Problem: Possible Bipartition
* Difficulty: Medium
* 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
*/

bool possibleBipartition(int n, int** dislikes, int dislikesSize, int*
dislikesColSize) {

}
```

Go Solution:

```
// Problem: Possible Bipartition
// Difficulty: Medium
// 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 possibleBipartition(n int, dislikes [][]int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun possibleBipartition(n: Int, dislikes: Array<IntArray>): Boolean {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Possible Bipartition
// Difficulty: Medium
// 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 possible_bipartition(n: i32, dislikes: Vec<Vec<i32>>) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
bool possibleBipartition(int n, List<List<int>> dislikes) {

}
}
```

Scala Solution:

```
object Solution {
def possibleBipartition(n: Int, dislikes: Array[Array[Int]]): Boolean = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec possible_bipartition(N :: integer(), Dislikes :: [[integer()]]) ->
boolean().
possible_bipartition(N, Dislikes) ->
.
```

Racket Solution:

```
(define/contract (possible-bipartition n dislikes)
(-> exact-integer? (listof (listof exact-integer?)) boolean?)
)
```