Problem 785: Is Graph Bipartite?

Problem Information

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

Problem Description

There is an

undirected

graph with

n

nodes, where each node is numbered between

0

and

n - 1

. You are given a 2D array

graph

, where

graph[u]

is an array of nodes that node

u

is adjacent to. More formally, for each

v

in

graph[u]

, there is an undirected edge between node

u

and node

v

. The graph has the following properties:

There are no self-edges (

graph[u]

does not contain

u

).

There are no parallel edges (

graph[u]

does not contain duplicate values).

If

v

is in

graph[u]

, then

u

is in

graph[v]

(the graph is undirected).

The graph may not be connected, meaning there may be two nodes

u

and

v

such that there is no path between them.

A graph is

bipartite

if the nodes can be partitioned into two independent sets

A

and

B

such that

every

edge in the graph connects a node in set

A

and a node in set

B

.

Return

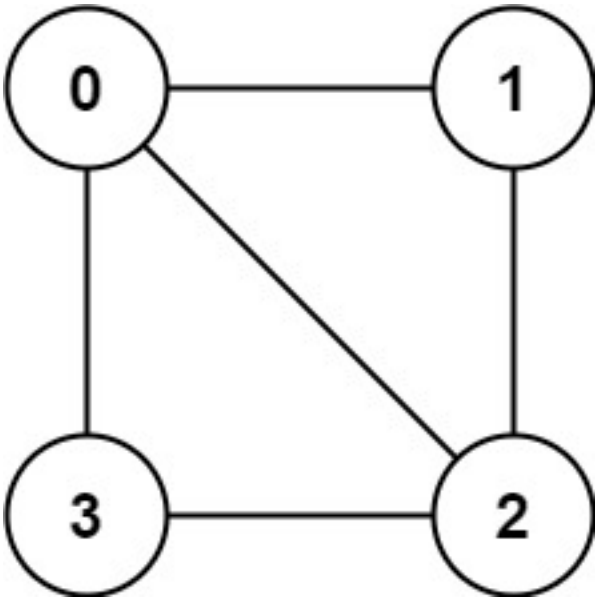
true

if and only if it is

bipartite

.

Example 1:



Input:

graph = [[1,2,3],[0,2],[0,1,3],[0,2]]

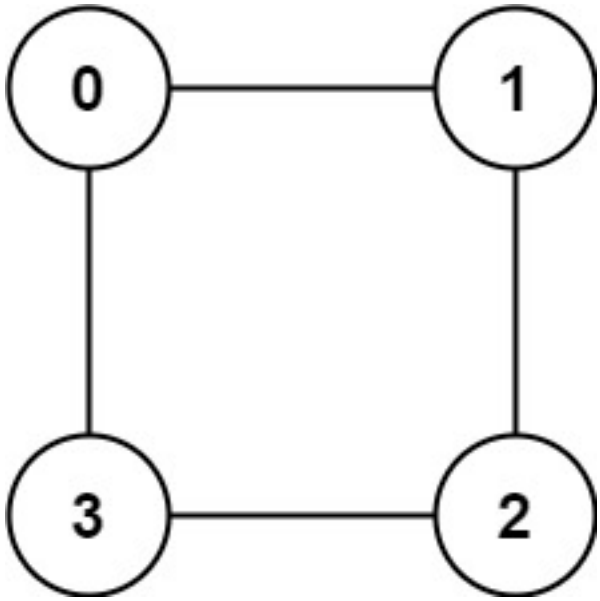
Output:

false

Explanation:

There is no way to partition the nodes into two independent sets such that every edge connects a node in one and a node in the other.

Example 2:



Input:

graph = [[1,3],[0,2],[1,3],[0,2]]

Output:

true

Explanation:

We can partition the nodes into two sets: {0, 2} and {1, 3}.

Constraints:

graph.length == n

1 <= n <= 100

0 <= graph[u].length < n

0 <= graph[u][i] <= n - 1

graph[u]

does not contain

u

.

All the values of

graph[u]

are

unique

.

If

graph[u]

contains

v

, then

graph[v]

contains

u

.

Code Snippets

C++:

```
class Solution {
public:
bool isBipartite(vector<vector<int>>& graph) {

}
};
```

Java:

```
class Solution {
public boolean isBipartite(int[][] graph) {

}
}
```

Python3:

```
class Solution:
def isBipartite(self, graph: List[List[int]]) -> bool:
```

Python:

```
class Solution(object):
def isBipartite(self, graph):
"""
:type graph: List[List[int]]
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {number[][]} graph
* @return {boolean}
*/
var isBipartite = function(graph) {

};
```

TypeScript:

```
function isBipartite(graph: number[][]): boolean {

};
```

C#:

```
public class Solution {
public bool IsBipartite(int[][] graph) {

}
}
```

C:

```
bool isBipartite(int** graph, int graphSize, int* graphColSize) {

}
```

Go:

```
func isBipartite(graph [][]int) bool {

}
```

Kotlin:

```
class Solution {
fun isBipartite(graph: Array<IntArray>): Boolean {

}
}
```

Swift:

```
class Solution {
func isBipartite(_ graph: [[Int]]) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn is_bipartite(graph: Vec<Vec<i32>>) -> bool {

}
}
```

Ruby:

```
# @param {Integer[][]} graph
# @return {Boolean}
def is_bipartite(graph)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $graph
* @return Boolean
*/
function isBipartite($graph) {

}
}
```

Dart:

```
class Solution {
bool isBipartite(List<List<int>> graph) {

}
}
```

Scala:

```
object Solution {
def isBipartite(graph: Array[Array[Int]]): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec is_bipartite(graph :: [[integer]]) :: boolean
def is_bipartite(graph) do

end
end
```

Erlang:

```
-spec is_bipartite(Graph :: [[integer()]]) -> boolean().
is_bipartite(Graph) ->
.
```

Racket:

```
(define/contract (is-bipartite graph)
(-> (listof (listof exact-integer?)) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Is Graph Bipartite?
* 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 isBipartite(vector<vector<int>>& graph) {

}
};
```

Java Solution:

```
/**
* Problem: Is Graph Bipartite?
* 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 isBipartite(int[][] graph) {

}
}
```

Python3 Solution:

```
"""
Problem: Is Graph Bipartite?
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 isBipartite(self, graph: List[List[int]]) -> bool:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class Solution(object):
def isBipartite(self, graph):
"""
:type graph: List[List[int]]
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Is Graph Bipartite?
* 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[][]} graph
* @return {boolean}
*/
var isBipartite = function(graph) {

};
```

TypeScript Solution:

```
/**
* Problem: Is Graph Bipartite?
* 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 isBipartite(graph: number[][]): boolean {

};
```

C# Solution:

```
/*
* Problem: Is Graph Bipartite?
* 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 IsBipartite(int[][] graph) {

}
}
```

C Solution:

```
/*
* Problem: Is Graph Bipartite?
* 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 isBipartite(int** graph, int graphSize, int* graphColSize) {

}
```

Go Solution:

```
// Problem: Is Graph Bipartite?
// 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 isBipartite(graph [][]int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun isBipartite(graph: Array<IntArray>): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func isBipartite(_ graph: [[Int]]) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Is Graph Bipartite?
// 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 is_bipartite(graph: Vec<Vec<i32>>) -> bool {

}
```

```
}
```

Ruby Solution:

```
# @param {Integer[][]} graph
# @return {Boolean}
def is_bipartite(graph)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $graph
* @return Boolean
*/
function isBipartite($graph) {

}
}
```

Dart Solution:

```
class Solution {
bool isBipartite(List<List<int>> graph) {

}
}
```

Scala Solution:

```
object Solution {
def isBipartite(graph: Array[Array[Int]]): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec is_bipartite(graph :: [[integer]]) :: boolean
def is_bipartite(graph) do

end
end
```

Erlang Solution:

```
-spec is_bipartite(Graph :: [[integer()]]) -> boolean().
is_bipartite(Graph) ->
.
```

Racket Solution:

```
(define/contract (is-bipartite graph)
(-> (listof (listof exact-integer?)) boolean?)
)
```