Problem 1136: Parallel Courses

Problem Information

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

Problem Description

You are given an integer

n

, which indicates that there are

n

courses labeled from

1

to

n

. You are also given an array

relations

where

relations[i] = [prevCourse

i

, nextCourse

i

]

, representing a prerequisite relationship between course

prevCourse

i

and course

nextCourse

i

: course

prevCourse

i

has to be taken before course

nextCourse

i

.

In one semester, you can take

any number

of courses as long as you have taken all the prerequisites in the

previous

semester for the courses you are taking.

Return

the

minimum

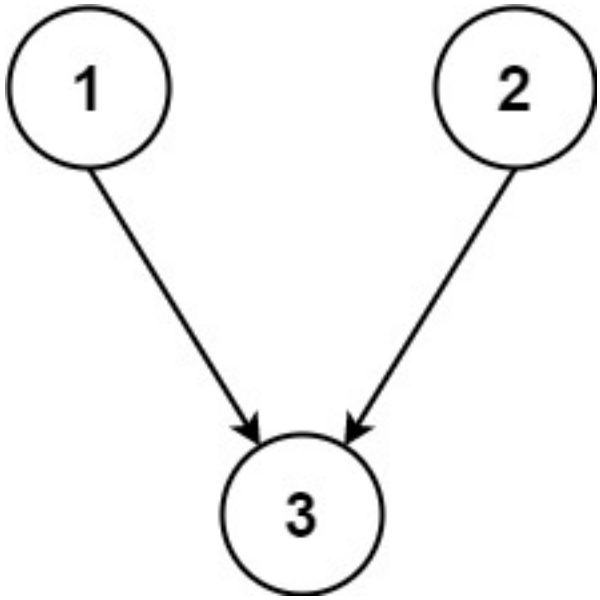
number of semesters needed to take all courses

. If there is no way to take all the courses, return

-1

.

Example 1:



Input:

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

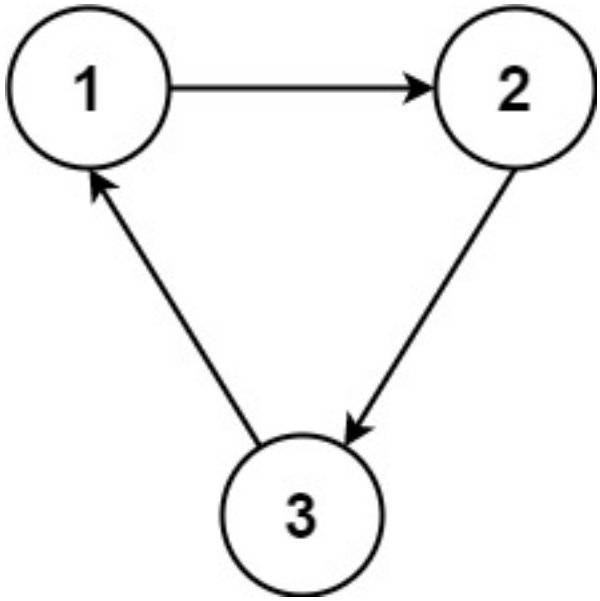
Output:

2

Explanation:

The figure above represents the given graph. In the first semester, you can take courses 1 and 2. In the second semester, you can take course 3.

Example 2:



Input:

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

Output:

-1

Explanation:

No course can be studied because they are prerequisites of each other.

Constraints:

1 <= n <= 5000

1 <= relations.length <= 5000

relations[i].length == 2

1 <= prevCourse

i

, nextCourse

i

<= n

prevCourse

i

!= nextCourse

i

All the pairs

[prevCourse

i

, nextCourse

i

]

are

unique

.

Code Snippets

C++:

```
class Solution {
public:
int minimumSemesters(int n, vector<vector<int>>& relations) {

}
};
```

Java:

```
class Solution {
public int minimumSemesters(int n, int[][] relations) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

```
};
```

TypeScript:

```
function minimumSemesters(n: number, relations: number[][]): number {

};
```

C#:

```
public class Solution {
public int MinimumSemesters(int n, int[][] relations) {

}
}
```

C:

```
int minimumSemesters(int n, int** relations, int relationsSize, int*
relationsColSize) {

}
```

Go:

```
func minimumSemesters(n int, relations [][]int) int {

}
```

Kotlin:

```
class Solution {
fun minimumSemesters(n: Int, relations: Array<IntArray>): Int {

}
}
```

Swift:

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

```
}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} relations
# @return {Integer}
def minimum_semesters(n, relations)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $relations
* @return Integer
*/
function minimumSemesters($n, $relations) {

}
}
```

Dart:

```
class Solution {
int minimumSemesters(int n, List<List<int>> relations) {

}
}
```

Scala:

```
object Solution {
def minimumSemesters(n: Int, relations: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_semesters(n :: integer, relations :: [[integer]]) :: integer
def minimum_semesters(n, relations) do

end
end
```

Erlang:

```
-spec minimum_semesters(N :: integer(), Relations :: [[integer()]]) ->
integer().
minimum_semesters(N, Relations) ->
.
```

Racket:

```
(define/contract (minimum-semesters n relations)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Parallel Courses
* Difficulty: Medium
* Tags: array, graph, sort
*
* 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 minimumSemesters(int n, vector<vector<int>>& relations) {

}
};
```

Java Solution:

```
/**
* Problem: Parallel Courses
* Difficulty: Medium
* Tags: array, graph, sort
*
* 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 minimumSemesters(int n, int[][] relations) {

}
}
```

Python3 Solution:

```
"""
Problem: Parallel Courses
Difficulty: Medium
Tags: array, graph, sort

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 minimumSemesters(self, n: int, relations: List[List[int]]) -> int:
```

```
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Parallel Courses
* Difficulty: Medium
* Tags: array, graph, sort
*
* 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[][]} relations
* @return {number}
*/
var minimumSemesters = function(n, relations) {

};
```

TypeScript Solution:

```
/**
* Problem: Parallel Courses
* Difficulty: Medium
* Tags: array, graph, sort
*
```

```
* 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 minimumSemesters(n: number, relations: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Parallel Courses
* Difficulty: Medium
* Tags: array, graph, sort
*
* 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 MinimumSemesters(int n, int[][] relations) {

}
}
```

C Solution:

```
/*
* Problem: Parallel Courses
* Difficulty: Medium
* Tags: array, graph, sort
*
* 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 minimumSemesters(int n, int** relations, int relationsSize, int*
relationsColSize) {
```

```
}
```

Go Solution:

```
// Problem: Parallel Courses
// Difficulty: Medium
// Tags: array, graph, sort
//
// 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 minimumSemesters(n int, relations [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumSemesters(n: Int, relations: Array<IntArray>): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Parallel Courses
// Difficulty: Medium
// Tags: array, graph, sort
//
// 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 minimum_semesters(n: i32, relations: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} relations
# @return {Integer}
def minimum_semesters(n, relations)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $relations
* @return Integer
*/
function minimumSemesters($n, $relations) {

}
}
```

Dart Solution:

```
class Solution {
int minimumSemesters(int n, List<List<int>> relations) {

}
}
```

Scala Solution:

```
object Solution {
def minimumSemesters(n: Int, relations: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_semesters(n :: integer, relations :: [[integer]]) :: integer
def minimum_semesters(n, relations) do

end
end
```

Erlang Solution:

```
-spec minimum_semesters(N :: integer(), Relations :: [[integer()]]) ->
integer().
minimum_semesters(N, Relations) ->
.
```

Racket Solution:

```
(define/contract (minimum-semesters n relations)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```