Problem 207: Course Schedule

Problem Information

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

Problem Description

There are a total of

numCourses

courses you have to take, labeled from

0

to

numCourses - 1

. You are given an array

prerequisites

where

prerequisites[i] = [a

i

, b

i

]

indicates that you

must

take course

b

i

first if you want to take course

a

i

.

For example, the pair

[0, 1]

, indicates that to take course

0

you have to first take course

1

.

Return

true

if you can finish all courses. Otherwise, return false

.

Example 1:

Input:

numCourses = 2, prerequisites = [[1,0]]

Output:

true

Explanation:

There are a total of 2 courses to take. To take course 1 you should have finished course 0. So it is possible.

Example 2:

Input:

numCourses = 2, prerequisites = [[1,0],[0,1]]

Output:

false

Explanation:

There are a total of 2 courses to take. To take course 1 you should have finished course 0, and to take course 0 you should also have finished course 1. So it is impossible.

Constraints:

1 <= numCourses <= 2000

0 <= prerequisites.length <= 5000

prerequisites[i].length == 2

0 <= a

i

, b

i

< numCourses

All the pairs prerequisites[i] are

unique

.

Code Snippets

C++:

```
class Solution {
public:
bool canFinish(int numCourses, vector<vector<int>>& prerequisites) {

}
};
```

Java:

```
class Solution {
public boolean canFinish(int numCourses, int[][] prerequisites) {

}
}
```

Python3:

```
class Solution:
def canFinish(self, numCourses: int, prerequisites: List[List[int]]) -> bool:
```

Python:

```
class Solution(object):
def canFinish(self, numCourses, prerequisites):
"""
:type numCourses: int
:type prerequisites: List[List[int]]
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {number} numCourses
* @param {number[][]} prerequisites
* @return {boolean}
*/
var canFinish = function(numCourses, prerequisites) {

};
```

TypeScript:

```
function canFinish(numCourses: number, prerequisites: number[][]): boolean {

};
```

C#:

```
public class Solution {
public bool CanFinish(int numCourses, int[][] prerequisites) {

}
}
```

C:

```
bool canFinish(int numCourses, int** prerequisites, int prerequisitesSize,
int* prerequisitesColSize) {
}
```

Go:

```
func canFinish(numCourses int, prerequisites [][]int) bool {

}
```

Kotlin:

```
class Solution {
fun canFinish(numCourses: Int, prerequisites: Array<IntArray>): Boolean {

}
}
```

Swift:

```
class Solution {
func canFinish(_ numCourses: Int, _ prerequisites: [[Int]]) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn can_finish(num_courses: i32, prerequisites: Vec<Vec<i32>>) -> bool {

}
}
```

Ruby:

```
# @param {Integer} num_courses
# @param {Integer[][]} prerequisites
# @return {Boolean}
def can_finish(num_courses, prerequisites)

end
```

PHP:

```
class Solution {

/**
* @param Integer $numCourses
* @param Integer[][] $prerequisites
* @return Boolean
*/
function canFinish($numCourses, $prerequisites) {

}
}
```

Dart:

```
class Solution {
bool canFinish(int numCourses, List<List<int>> prerequisites) {

}
}
```

Scala:

```
object Solution {
def canFinish(numCourses: Int, prerequisites: Array[Array[Int]]): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec can_finish(num_courses :: integer, prerequisites :: [[integer]]) ::
boolean
def can_finish(num_courses, prerequisites) do

end
end
```

Erlang:

```
-spec can_finish(NumCourses :: integer(), Prerequisites :: [[integer()]]) ->
boolean().
can_finish(NumCourses, Prerequisites) ->
```

```
.
```

Racket:

```
(define/contract (can-finish numCourses prerequisites)
(-> exact-integer? (listof (listof exact-integer?)) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Course Schedule
* Difficulty: Medium
* Tags: array, graph, sort, 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 canFinish(int numCourses, vector<vector<int>>& prerequisites) {

}
};
```

Java Solution:

```
/**
* Problem: Course Schedule
* Difficulty: Medium
* Tags: array, graph, sort, 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 canFinish(int numCourses, int[][] prerequisites) {

}
}
```

Python3 Solution:

```
"""
Problem: Course Schedule
Difficulty: Medium
Tags: array, graph, sort, 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 canFinish(self, numCourses: int, prerequisites: List[List[int]]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def canFinish(self, numCourses, prerequisites):
"""
:type numCourses: int
:type prerequisites: List[List[int]]
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Course Schedule
* Difficulty: Medium
* Tags: array, graph, sort, 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} numCourses
* @param {number[][]} prerequisites
* @return {boolean}
*/
var canFinish = function(numCourses, prerequisites) {

};
```

TypeScript Solution:

```
/**
* Problem: Course Schedule
* Difficulty: Medium
* Tags: array, graph, sort, 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 canFinish(numCourses: number, prerequisites: number[][]): boolean {

};
```

C# Solution:

```
/*
* Problem: Course Schedule
* Difficulty: Medium
* Tags: array, graph, sort, 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 CanFinish(int numCourses, int[][] prerequisites) {

}
}
```

C Solution:

```
/*
* Problem: Course Schedule
* Difficulty: Medium
* Tags: array, graph, sort, 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 canFinish(int numCourses, int** prerequisites, int prerequisitesSize,
int* prerequisitesColSize) {

}
```

Go Solution:

```
// Problem: Course Schedule
// Difficulty: Medium
// Tags: array, graph, sort, 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 canFinish(numCourses int, prerequisites [][]int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun canFinish(numCourses: Int, prerequisites: Array<IntArray>): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func canFinish(_ numCourses: Int, _ prerequisites: [[Int]]) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Course Schedule
// Difficulty: Medium
// Tags: array, graph, sort, 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 can_finish(num_courses: i32, prerequisites: Vec<Vec<i32>>) -> bool {

}
}
```

Ruby Solution:

```
# @param {Integer} num_courses
# @param {Integer[][]} prerequisites
# @return {Boolean}
def can_finish(num_courses, prerequisites)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $numCourses
* @param Integer[][] $prerequisites
* @return Boolean
*/
function canFinish($numCourses, $prerequisites) {

}
}
```

Dart Solution:

```
class Solution {
bool canFinish(int numCourses, List<List<int>> prerequisites) {

}
}
```

Scala Solution:

```
object Solution {
def canFinish(numCourses: Int, prerequisites: Array[Array[Int]]): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec can_finish(num_courses :: integer, prerequisites :: [[integer]]) ::
boolean
def can_finish(num_courses, prerequisites) do

end
end
```

Erlang Solution:

```
-spec can_finish(NumCourses :: integer(), Prerequisites :: [[integer()]]) ->
boolean().
```

```
can_finish(NumCourses, Prerequisites) ->
.
```

Racket Solution:

```
(define/contract (can-finish numCourses prerequisites)
(-> exact-integer? (listof (listof exact-integer?)) boolean?)
)
```