Problem 210: Course Schedule II

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

the ordering of courses you should take to finish all courses

. If there are many valid answers, return

any

of them. If it is impossible to finish all courses, return

an empty array

.

Example 1:

Input:

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

Output:

[0,1]

Explanation:

There are a total of 2 courses to take. To take course 1 you should have finished course 0. So the correct course order is [0,1].

Example 2:

Input:

numCourses = 4, prerequisites = [[1,0],[2,0],[3,1],[3,2]]

Output:

[0,2,1,3]

Explanation:

There are a total of 4 courses to take. To take course 3 you should have finished both courses 1 and 2. Both courses 1 and 2 should be taken after you finished course 0. So one correct course order is [0,1,2,3]. Another correct ordering is [0,2,1,3].

Example 3:

Input:

numCourses = 1, prerequisites = []

Output:

[0]

Constraints:

1 <= numCourses <= 2000

0 <= prerequisites.length <= numCourses * (numCourses - 1)

prerequisites[i].length == 2

0 <= a

i

, b

i

< numCourses

a

i

!= b

i

All the pairs

[a

i

, b

i

]

are

distinct

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* findOrder(int numCourses, int** prerequisites, int prerequisitesSize,
int* prerequisitesColSize, int* returnSize) {
```

```
}
```

Go:

```
func findOrder(numCourses int, prerequisites [][]int) []int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (find-order numCourses prerequisites)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Course Schedule II
* 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:
vector<int> findOrder(int numCourses, vector<vector<int>>& prerequisites) {

}
};
```

Java Solution:

```
/**
* Problem: Course Schedule II
* 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 int[] findOrder(int numCourses, int[][] prerequisites) {

}
}
```

Python3 Solution:

```
"""
Problem: Course Schedule II
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 findOrder(self, numCourses: int, prerequisites: List[List[int]]) ->
List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Course Schedule II
```

```
* 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 {number[]}
*/
var findOrder = function(numCourses, prerequisites) {

};
```

TypeScript Solution:

```
/**
* Problem: Course Schedule II
* 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 findOrder(numCourses: number, prerequisites: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Course Schedule II
* 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 int[] FindOrder(int numCourses, int[][] prerequisites) {

}
}
```

C Solution:

```
/*
* Problem: Course Schedule II
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* findOrder(int numCourses, int** prerequisites, int prerequisitesSize,
int* prerequisitesColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Course Schedule II
// 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 findOrder(numCourses int, prerequisites [][]int) []int {
```

```
}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Course Schedule II
// 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 find_order(num_courses: i32, prerequisites: Vec<Vec<i32>>) -> Vec<i32>
{

}
}
```

Ruby Solution:

```
# @param {Integer} num_courses
# @param {Integer[][]} prerequisites
# @return {Integer[]}
```

```
def find_order(num_courses, prerequisites)

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

```
end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (find-order numCourses prerequisites)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?))
)
```