Problem 630: Course Schedule III

Problem Information

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

Problem Description

There are

n

different online courses numbered from

1

to

n

. You are given an array

courses

where

courses[i] = [duration

i

, lastDay

i

]

indicate that the

i

th

course should be taken

continuously

for

duration

i

days and must be finished before or on

lastDay

i

.

You will start on the

1

st

day and you cannot take two or more courses simultaneously.

Return

the maximum number of courses that you can take

.

Example 1:

Input:

courses = [[100,200],[200,1300],[1000,1250],[2000,3200]]

Output:

3 Explanation: There are totally 4 courses, but you can take 3 courses at most: First, take the 1

st

course, it costs 100 days so you will finish it on the 100

th

day, and ready to take the next course on the 101

st

day. Second, take the 3

rd

course, it costs 1000 days so you will finish it on the 1100

th

day, and ready to take the next course on the 1101

st

day. Third, take the 2

nd

course, it costs 200 days so you will finish it on the 1300

th

day. The 4

th

course cannot be taken now, since you will finish it on the 3300

th

day, which exceeds the closed date.

Example 2:

Input:

courses = [[1,2]]

Output:

1

Example 3:

Input:

courses = [[3,2],[4,3]]

Output:

0

Constraints:

1 <= courses.length <= 10

4

1 <= duration

i

, lastDay

i

<= 10

4

Code Snippets

C++:

```
class Solution {
public:
int scheduleCourse(vector<vector<int>>& courses) {

}
};
```

Java:

```
class Solution {
public int scheduleCourse(int[][] courses) {

}
}
```

Python3:

```
class Solution:
def scheduleCourse(self, courses: List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def scheduleCourse(self, courses):
```

```
"""
:type courses: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} courses
* @return {number}
*/
var scheduleCourse = function(courses) {

};
```

TypeScript:

```
function scheduleCourse(courses: number[][]): number {

};
```

C#:

```
public class Solution {
public int ScheduleCourse(int[][] courses) {

}
}
```

C:

```
int scheduleCourse(int** courses, int coursesSize, int* coursesColSize) {

}
```

Go:

```
func scheduleCourse(courses [][]int) int {

}
```

Kotlin:

```
class Solution {
fun scheduleCourse(courses: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func scheduleCourse(_ courses: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn schedule_course(courses: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} courses
# @return {Integer}
def schedule_course(courses)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $courses
* @return Integer
*/
function scheduleCourse($courses) {

}
}
```

Dart:

```
class Solution {
int scheduleCourse(List<List<int>> courses) {

}
}
```

Scala:

```
object Solution {
def scheduleCourse(courses: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec schedule_course(courses :: [[integer]]) :: integer
def schedule_course(courses) do

end
end
```

Erlang:

```
-spec schedule_course(Courses :: [[integer()]]) -> integer().
schedule_course(Courses) ->
.
```

Racket:

```
(define/contract (schedule-course courses)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Course Schedule III
* Difficulty: Hard
* Tags: array, greedy, sort, queue, heap
*
* 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 scheduleCourse(vector<vector<int>>& courses) {

}
};
```

Java Solution:

```
/**
* Problem: Course Schedule III
* Difficulty: Hard
* Tags: array, greedy, sort, queue, heap
*
* 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 scheduleCourse(int[][] courses) {

}
}
```

Python3 Solution:

```
"""
Problem: Course Schedule III
Difficulty: Hard
Tags: array, greedy, sort, queue, heap
```

```
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 scheduleCourse(self, courses: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def scheduleCourse(self, courses):
"""
:type courses: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Course Schedule III
* Difficulty: Hard
* Tags: array, greedy, sort, queue, heap
*
* 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[][]} courses
* @return {number}
*/
var scheduleCourse = function(courses) {

};
```

TypeScript Solution:

```
/**
* Problem: Course Schedule III
* Difficulty: Hard
* Tags: array, greedy, sort, queue, heap
*
* 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 scheduleCourse(courses: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Course Schedule III
* Difficulty: Hard
* Tags: array, greedy, sort, queue, heap
*
* 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 ScheduleCourse(int[][] courses) {

}
}
```

C Solution:

```
/*
* Problem: Course Schedule III
* Difficulty: Hard
* Tags: array, greedy, sort, queue, heap
*
* 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 scheduleCourse(int** courses, int coursesSize, int* coursesColSize) {

}
```

Go Solution:

```
// Problem: Course Schedule III
// Difficulty: Hard
// Tags: array, greedy, sort, queue, heap
//
// 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 scheduleCourse(courses [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun scheduleCourse(courses: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func scheduleCourse(_ courses: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Course Schedule III
// Difficulty: Hard
// Tags: array, greedy, sort, queue, heap
```

```
//
// 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 schedule_course(courses: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} courses
# @return {Integer}
def schedule_course(courses)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $courses
* @return Integer
*/
function scheduleCourse($courses) {

}
}
```

Dart Solution:

```
class Solution {
int scheduleCourse(List<List<int>> courses) {

}
}
```

Scala Solution:

```
object Solution {
def scheduleCourse(courses: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec schedule_course(courses :: [[integer]]) :: integer
def schedule_course(courses) do

end
end
```

Erlang Solution:

```
-spec schedule_course(Courses :: [[integer()]]) -> integer().
schedule_course(Courses) ->
.
```

Racket Solution:

```
(define/contract (schedule-course courses)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```