Problem 1353: Maximum Number of Events That Can Be Attended

Problem Information

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

Problem Description

You are given an array of

events

where

events[i] = [startDay

i

, endDay

i

]

. Every event

i

starts at

startDay

i

and ends at

endDay

i

.

You can attend an event

i

at any day

d

where

startDay

i

<= d <= endDay

i

. You can only attend one event at any time

d

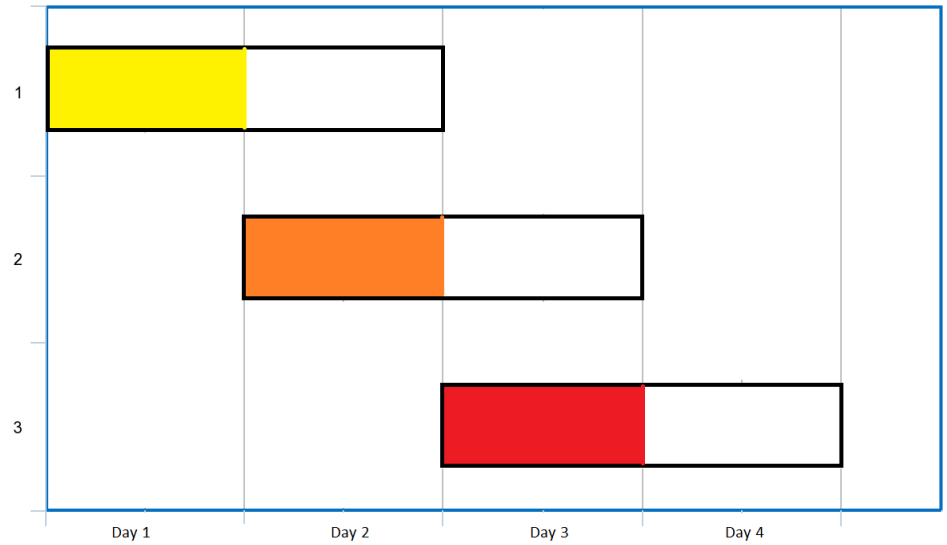
.

Return

the maximum number of events you can attend

.

Example 1:



Input:

events = [[1,2],[2,3],[3,4]]

Output:

3

Explanation:

You can attend all the three events. One way to attend them all is as shown. Attend the first event on day 1. Attend the second event on day 2. Attend the third event on day 3.

Example 2:

Input:

events= [[1,2],[2,3],[3,4],[1,2]]

Output:

4

Constraints:

1 <= events.length <= 10

5

events[i].length == 2

1 <= startDay

i

<= endDay

i

<= 10

5

Code Snippets

C++:

```
class Solution {
public:
int maxEvents(vector<vector<int>>& events) {

}
};
```

Java:

```
class Solution {
public int maxEvents(int[][] events) {

}
}
```

Python3:

```
class Solution:
def maxEvents(self, events: List[List[int]]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maxEvents(events: number[][]): number {

};
```

C#:

```
public class Solution {
public int MaxEvents(int[][] events) {

}
}
```

C:

```
int maxEvents(int** events, int eventsSize, int* eventsColSize) {

}
```

Go:

```
func maxEvents(events [][]int) int {

}
```

Kotlin:

```
class Solution {
fun maxEvents(events: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func maxEvents(_ events: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_events(events: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} events
# @return {Integer}
def max_events(events)

end
```

PHP:

```
class Solution {

/**
```

```
* @param Integer[][] $events
* @return Integer
*/
function maxEvents($events) {

}
}
```

Dart:

```
class Solution {
int maxEvents(List<List<int>> events) {

}
}
```

Scala:

```
object Solution {
def maxEvents(events: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_events(events :: [[integer]]) :: integer
def max_events(events) do

end
end
```

Erlang:

```
-spec max_events(Events :: [[integer()]]) -> integer().
max_events(Events) ->
.
```

Racket:

```
(define/contract (max-events events)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Number of Events That Can Be Attended
* Difficulty: Medium
* 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 maxEvents(vector<vector<int>>& events) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Number of Events That Can Be Attended
* Difficulty: Medium
* 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 maxEvents(int[][] events) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Maximum Number of Events That Can Be Attended
Difficulty: Medium
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 maxEvents(self, events: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Number of Events That Can Be Attended
* Difficulty: Medium
* 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[][]} events
* @return {number}
*/
var maxEvents = function(events) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Number of Events That Can Be Attended
* Difficulty: Medium
* 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 maxEvents(events: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Number of Events That Can Be Attended
* Difficulty: Medium
* 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 MaxEvents(int[][] events) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Number of Events That Can Be Attended
* Difficulty: Medium
* 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 maxEvents(int** events, int eventsSize, int* eventsColSize) {

}
```

Go Solution:

```
// Problem: Maximum Number of Events That Can Be Attended
// Difficulty: Medium
// 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 maxEvents(events [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxEvents(events: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxEvents(_ events: [[Int]]) -> Int {
```

```
}
}
```

Rust Solution:

```
// Problem: Maximum Number of Events That Can Be Attended
// Difficulty: Medium
// 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 max_events(events: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} events
# @return {Integer}
def max_events(events)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $events
* @return Integer
*/
function maxEvents($events) {

}
}
```

Dart Solution:

```
class Solution {
int maxEvents(List<List<int>> events) {

}
}
```

Scala Solution:

```
object Solution {
def maxEvents(events: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_events(events :: [[integer]]) :: integer
def max_events(events) do

end
end
```

Erlang Solution:

```
-spec max_events(Events :: [[integer()]]) -> integer().
max_events(Events) ->
.
```

Racket Solution:

```
(define/contract (max-events events)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```