Problem 729: My Calendar I

Problem Information

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

Problem Description

You are implementing a program to use as your calendar. We can add a new event if adding the event will not cause a

double booking

.

A

double booking

happens when two events have some non-empty intersection (i.e., some moment is common to both events.).

The event can be represented as a pair of integers

startTime

and

endTime

that represents a booking on the half-open interval

[startTime, endTime)

, the range of real numbers

x

such that

startTime <= x < endTime

.

Implement the

MyCalendar

class:

MyCalendar()

Initializes the calendar object.

boolean book(int startTime, int endTime)

Returns

true

if the event can be added to the calendar successfully without causing a

double booking

. Otherwise, return

false

and do not add the event to the calendar.

Example 1:

Input

["MyCalendar", "book", "book", "book"] [[], [10, 20], [15, 25], [20, 30]]

Output

[null, true, false, true]

Explanation

MyCalendar myCalendar = new MyCalendar(); myCalendar.book(10, 20); // return True myCalendar.book(15, 25); // return False, It can not be booked because time 15 is already booked by another event. myCalendar.book(20, 30); // return True, The event can be booked, as the first event takes every time less than 20, but not including 20.

Constraints:

0 <= start < end <= 10

9

At most

1000

calls will be made to

book

.

Code Snippets

C++:

```
class MyCalendar {
public:
MyCalendar() {

}
```

```
bool book(int startTime, int endTime) {

}
};

/**
* Your MyCalendar object will be instantiated and called as such:
* MyCalendar* obj = new MyCalendar();
* bool param_1 = obj->book(startTime,endTime);
*/
```

Java:

```
class MyCalendar {

public MyCalendar() {

}

public boolean book(int startTime, int endTime) {

}
}

/**
* Your MyCalendar object will be instantiated and called as such:
* MyCalendar obj = new MyCalendar();
* boolean param_1 = obj.book(startTime,endTime);
*/
```

Python3:

```
class MyCalendar:

def __init__(self):


def book(self, startTime: int, endTime: int) -> bool:
```

```
# Your MyCalendar object will be instantiated and called as such:
# obj = MyCalendar()
# param_1 = obj.book(startTime,endTime)
```

Python:

```
class MyCalendar(object):

def __init__(self):


def book(self, startTime, endTime):
"""
:type startTime: int
:type endTime: int
:rtype: bool
"""



# Your MyCalendar object will be instantiated and called as such:
# obj = MyCalendar()
# param_1 = obj.book(startTime,endTime)
```

JavaScript:

```
var MyCalendar = function() {

};

/**
* @param {number} startTime
* @param {number} endTime
* @return {boolean}
*/
MyCalendar.prototype.book = function(startTime, endTime) {

};

/**
* Your MyCalendar object will be instantiated and called as such:
```

```
* var obj = new MyCalendar()
* var param_1 = obj.book(startTime,endTime)
*/
```

TypeScript:

```
class MyCalendar {
constructor() {

}

book(startTime: number, endTime: number): boolean {

}
}

/**
* Your MyCalendar object will be instantiated and called as such:
* var obj = new MyCalendar()
* var param_1 = obj.book(startTime,endTime)
*/
```

C#:

```
public class MyCalendar {

public MyCalendar() {

}

public bool Book(int startTime, int endTime) {

}
}

/**
* Your MyCalendar object will be instantiated and called as such:
* MyCalendar obj = new MyCalendar();
* bool param_1 = obj.Book(startTime,endTime);
*/
```

C:

```
typedef struct {

} MyCalendar;


MyCalendar* myCalendarCreate() {

}

bool myCalendarBook(MyCalendar* obj, int startTime, int endTime) {

}

void myCalendarFree(MyCalendar* obj) {

}

/**
* Your MyCalendar struct will be instantiated and called as such:
* MyCalendar* obj = myCalendarCreate();
* bool param_1 = myCalendarBook(obj, startTime, endTime);

* myCalendarFree(obj);
*/
```

Go:

```
type MyCalendar struct {

}


func Constructor() MyCalendar {

}


func (this *MyCalendar) Book(startTime int, endTime int) bool {

}
```

```
/**
* Your MyCalendar object will be instantiated and called as such:
* obj := Constructor();
* param_1 := obj.Book(startTime,endTime);
*/
```

Kotlin:

```
class MyCalendar() {

fun book(startTime: Int, endTime: Int): Boolean {

}

}

/**
* Your MyCalendar object will be instantiated and called as such:
* var obj = MyCalendar()
* var param_1 = obj.book(startTime,endTime)
*/
```

Swift:

```
class MyCalendar {

init() {

}

func book(_ startTime: Int, _ endTime: Int) -> Bool {

}
}

/**
* Your MyCalendar object will be instantiated and called as such:
* let obj = MyCalendar()
* let ret_1: Bool = obj.book(startTime, endTime)
```

```
*/
```

Rust:

```
struct MyCalendar {

}


/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl MyCalendar {

fn new() -> Self {

}

fn book(&self, start_time: i32, end_time: i32) -> bool {

}
}

/**
* Your MyCalendar object will be instantiated and called as such:
* let obj = MyCalendar::new();
* let ret_1: bool = obj.book(startTime, endTime);
*/
```

Ruby:

```
class MyCalendar
def initialize()

end


=begin
:type start_time: Integer
:type end_time: Integer
:rtype: Boolean
```

```
=end
def book(start_time, end_time)

end



end

# Your MyCalendar object will be instantiated and called as such:
# obj = MyCalendar.new()
# param_1 = obj.book(start_time, end_time)
```

PHP:

```
class MyCalendar {
/**
*/
function __construct() {

}

/**
* @param Integer $startTime
* @param Integer $endTime
* @return Boolean
*/
function book($startTime, $endTime) {

}
}

/**
* Your MyCalendar object will be instantiated and called as such:
* $obj = MyCalendar();
* $ret_1 = $obj->book($startTime, $endTime);
*/
```

Dart:

```
class MyCalendar {

MyCalendar() {
```

```
}

bool book(int startTime, int endTime) {

}
}

/**
* Your MyCalendar object will be instantiated and called as such:
* MyCalendar obj = MyCalendar();
* bool param1 = obj.book(startTime,endTime);
*/
```

Scala:

```
class MyCalendar() {

def book(startTime: Int, endTime: Int): Boolean = {

}

}

/**
* Your MyCalendar object will be instantiated and called as such:
* val obj = new MyCalendar()
* val param_1 = obj.book(startTime,endTime)
*/
```

Elixir:

```
defmodule MyCalendar do
@spec init_() :: any
def init_() do

end

@spec book(start_time :: integer, end_time :: integer) :: boolean
def book(start_time, end_time) do

end
```

```
end

# Your functions will be called as such:
# MyCalendar.init_()
# param_1 = MyCalendar.book(start_time, end_time)

# MyCalendar.init_ will be called before every test case, in which you can do some necessary initializations.
```

Erlang:

```
-spec my_calendar_init_() -> any().
my_calendar_init_() ->
.

-spec my_calendar_book(StartTime :: integer(), EndTime :: integer()) -> boolean().
my_calendar_book(StartTime, EndTime) ->
.


%% Your functions will be called as such:
%% my_calendar_init_(),
%% Param_1 = my_calendar_book(StartTime, EndTime),

%% my_calendar_init_ will be called before every test case, in which you can do some necessary initializations.
```

Racket:

```
(define my-calendar%
(class object%
(super-new)

(init-field)

; book : exact-integer? exact-integer? -> boolean?
(define/public (book start-time end-time)
)))

;; Your my-calendar% object will be instantiated and called as such:
;; (define obj (new my-calendar%))
```

```
;; (define param_1 (send obj book start-time end-time))
```

Solutions

C++ Solution:

```
/*
* Problem: My Calendar I
* Difficulty: Medium
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class MyCalendar {
public:
MyCalendar() {

}

bool book(int startTime, int endTime) {

}
};

/**
* Your MyCalendar object will be instantiated and called as such:
* MyCalendar* obj = new MyCalendar();
* bool param_1 = obj->book(startTime,endTime);
*/
```

Java Solution:

```
/**
* Problem: My Calendar I
* Difficulty: Medium
* Tags: array, tree, search
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class MyCalendar {

public MyCalendar() {

}

public boolean book(int startTime, int endTime) {

}
}

/**
* Your MyCalendar object will be instantiated and called as such:
* MyCalendar obj = new MyCalendar();
* boolean param_1 = obj.book(startTime,endTime);
*/
```

Python3 Solution:

```
"""
Problem: My Calendar I
Difficulty: Medium
Tags: array, tree, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class MyCalendar:

def __init__(self):


def book(self, startTime: int, endTime: int) -> bool:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class MyCalendar(object):

def __init__(self):


def book(self, startTime, endTime):
"""
:type startTime: int
:type endTime: int
:rtype: bool
"""



# Your MyCalendar object will be instantiated and called as such:
# obj = MyCalendar()
# param_1 = obj.book(startTime,endTime)
```

JavaScript Solution:

```
/**
* Problem: My Calendar I
* Difficulty: Medium
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/


var MyCalendar = function() {

};

/**
* @param {number} startTime
```

```
* @param {number} endTime
* @return {boolean}
*/
MyCalendar.prototype.book = function(startTime, endTime) {

};

/**
* Your MyCalendar object will be instantiated and called as such:
* var obj = new MyCalendar()
* var param_1 = obj.book(startTime,endTime)
*/
```

TypeScript Solution:

```
/**
* Problem: My Calendar I
* Difficulty: Medium
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class MyCalendar {
constructor() {

}

book(startTime: number, endTime: number): boolean {

}
}

/**
* Your MyCalendar object will be instantiated and called as such:
* var obj = new MyCalendar()
* var param_1 = obj.book(startTime,endTime)
*/
```

C# Solution:

```
/*
* Problem: My Calendar I
* Difficulty: Medium
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class MyCalendar {

public MyCalendar() {

}

public bool Book(int startTime, int endTime) {

}
}

/**
* Your MyCalendar object will be instantiated and called as such:
* MyCalendar obj = new MyCalendar();
* bool param_1 = obj.Book(startTime,endTime);
*/
```

C Solution:

```
/*
* Problem: My Calendar I
* Difficulty: Medium
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/
```

```
typedef struct {

} MyCalendar;


MyCalendar* myCalendarCreate() {

}

bool myCalendarBook(MyCalendar* obj, int startTime, int endTime) {

}

void myCalendarFree(MyCalendar* obj) {

}

/**
* Your MyCalendar struct will be instantiated and called as such:
* MyCalendar* obj = myCalendarCreate();
* bool param_1 = myCalendarBook(obj, startTime, endTime);

* myCalendarFree(obj);
*/
```

Go Solution:

```
// Problem: My Calendar I
// Difficulty: Medium
// Tags: array, tree, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

type MyCalendar struct {

}
```

```
func Constructor() MyCalendar {

}


func (this *MyCalendar) Book(startTime int, endTime int) bool {

}


/**
* Your MyCalendar object will be instantiated and called as such:
* obj := Constructor();
* param_1 := obj.Book(startTime,endTime);
*/
```

Kotlin Solution:

```
class MyCalendar() {

fun book(startTime: Int, endTime: Int): Boolean {

}

}

/**
* Your MyCalendar object will be instantiated and called as such:
* var obj = MyCalendar()
* var param_1 = obj.book(startTime,endTime)
*/
```

Swift Solution:

```
class MyCalendar {

init() {

}
```

```
func book(_ startTime: Int, _ endTime: Int) -> Bool {

}
}

/**
* Your MyCalendar object will be instantiated and called as such:
* let obj = MyCalendar()
* let ret_1: Bool = obj.book(startTime, endTime)
*/
```

Rust Solution:

```
// Problem: My Calendar I
// Difficulty: Medium
// Tags: array, tree, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

struct MyCalendar {

}



/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl MyCalendar {

fn new() -> Self {

}

fn book(&self, start_time: i32, end_time: i32) -> bool {

}
}
```

```
/**
* Your MyCalendar object will be instantiated and called as such:
* let obj = MyCalendar::new();
* let ret_1: bool = obj.book(startTime, endTime);
*/
```

Ruby Solution:

```
class MyCalendar
def initialize()

end


=begin
:type start_time: Integer
:type end_time: Integer
:rtype: Boolean
=end
def book(start_time, end_time)

end


end

# Your MyCalendar object will be instantiated and called as such:
# obj = MyCalendar.new()
# param_1 = obj.book(start_time, end_time)
```

PHP Solution:

```
class MyCalendar {
/**
*/
function __construct() {

}

/**
* @param Integer $startTime
```

```
* @param Integer $endTime
* @return Boolean
*/
function book($startTime, $endTime) {

}
}

/**
* Your MyCalendar object will be instantiated and called as such:
* $obj = MyCalendar();
* $ret_1 = $obj->book($startTime, $endTime);
*/
```

Dart Solution:

```
class MyCalendar {

MyCalendar() {

}

bool book(int startTime, int endTime) {

}
}

/**
* Your MyCalendar object will be instantiated and called as such:
* MyCalendar obj = MyCalendar();
* bool param1 = obj.book(startTime,endTime);
*/
```

Scala Solution:

```
class MyCalendar() {

def book(startTime: Int, endTime: Int): Boolean = {

}
```

```
}

/**
* Your MyCalendar object will be instantiated and called as such:
* val obj = new MyCalendar()
* val param_1 = obj.book(startTime,endTime)
*/
```

Elixir Solution:

```
defmodule MyCalendar do
@spec init_() :: any
def init_() do

end

@spec book(start_time :: integer, end_time :: integer) :: boolean
def book(start_time, end_time) do

end
end

# Your functions will be called as such:
# MyCalendar.init_()
# param_1 = MyCalendar.book(start_time, end_time)

# MyCalendar.init_ will be called before every test case, in which you can do
some necessary initializations.
```

Erlang Solution:

```
-spec my_calendar_init_() -> any().
my_calendar_init_() ->
.

-spec my_calendar_book(StartTime :: integer(), EndTime :: integer()) ->
boolean().
my_calendar_book(StartTime, EndTime) ->
.
```

```
%% Your functions will be called as such:
%% my_calendar_init_(),
%% Param_1 = my_calendar_book(StartTime, EndTime),

%% my_calendar_init_ will be called before every test case, in which you can
do some necessary initializations.
```

Racket Solution:

```
(define my-calendar%
(class object%
(super-new)

(init-field)

; book : exact-integer? exact-integer? -> boolean?
(define/public (book start-time end-time)
)))

;; Your my-calendar% object will be instantiated and called as such:
;; (define obj (new my-calendar%))
;; (define param_1 (send obj book start-time end-time))
```