Problem 759: Employee Free Time

Problem Information

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

Problem Description

We are given a list

schedule

of employees, which represents the working time for each employee.

Each employee has a list of non-overlapping

Intervals

, and these intervals are in sorted order.

Return the list of finite intervals representing

common, positive-length free time

for

all

employees, also in sorted order.

(Even though we are representing

Intervals

in the form

[x, y]

, the objects inside are

Intervals

, not lists or arrays. For example,

schedule[0][0].start = 1

,

schedule[0][0].end = 2

, and

schedule[0][0][0]

is not defined). Also, we wouldn't include intervals like [5, 5] in our answer, as they have zero length.

Example 1:

Input:

schedule = [[[1,2],[5,6]],[[1,3]],[[4,10]]]

Output:

[[3,4]]

Explanation:

There are a total of three employees, and all common free time intervals would be [-inf, 1], [3, 4], [10, inf]. We discard any intervals that contain inf as they aren't finite.

Example 2:

Input:

schedule = [[[1,3],[6,7]],[[2,4]],[[2,5],[9,12]]]

Output:

[[5,6],[7,9]]

Constraints:

1 <= schedule.length , schedule[i].length <= 50

0 <= schedule[i].start < schedule[i].end <= 10^8

Code Snippets

C++:

```
/*
// Definition for an Interval.
class Interval {
public:
int start;
int end;

Interval() {}

Interval(int _start, int _end) {
start = _start;
end = _end;
}
};
*/

class Solution {
public:
vector<Interval> employeeFreeTime(vector<vector<Interval>> schedule) {

}
};
```

Java:

```
/*
// Definition for an Interval.
class Interval {
public int start;
public int end;

public Interval() {}

public Interval(int _start, int _end) {
start = _start;
end = _end;
}
};
*/

class Solution {
public List<Interval> employeeFreeTime(List<List<Interval>> schedule) {

}
}
```

Python3:

```
"""
# Definition for an Interval.
class Interval:
def __init__(self, start: int = None, end: int = None):
self.start = start
self.end = end
"""

class Solution:
def employeeFreeTime(self, schedule: '[[Interval]]') -> '[Interval]':
```

Python:

```
"""
# Definition for an Interval.
class Interval(object):
def __init__(self, start=None, end=None):
self.start = start
self.end = end
"""

class Solution(object):
def employeeFreeTime(self, schedule):
"""
:type schedule: [[Interval]]
:rtype: [Interval]
"""
```

JavaScript:

```
/**
* // Definition for an Interval.
* function Interval(start, end) {
* this.start = start;
* this.end = end;
* };
*/

/**
* @param {Interval[][]} schedule
* @return {Interval[]}
*/
var employeeFreeTime = function(schedule) {

};
```

TypeScript:

```
/**
* // Definition for an Interval.
* class Interval {
* start: number;
* end: number;
* constructor(start: number, end: number) {
* this.start = start;
* this.end = end;
* }
* }
*/
```

```
function employeeFreeTime(schedule: Interval[][]): Interval[] {

};
```

C#:

```
/*
// Definition for an Interval.
public class Interval {
public int start;
public int end;

public Interval(){}
public Interval(int _start, int _end) {
start = _start;
end = _end;
}
}
*/

public class Solution {
public IList<Interval> EmployeeFreeTime(IList<IList<Interval>> schedule) {

}
}
```

Go:

```
/**
* Definition for an Interval.
* type Interval struct {
* Start int
* End int
* }
*/

func employeeFreeTime(schedule [][]*Interval) []*Interval {

}
```

Kotlin:

```
/*
* // Definition for an Interval.
* class Interval {
* var start:Int = 0
* var end:Int = 0
*
* constructor(_start:Int, _end:Int) {
* start = _start
* end = _end
* }
* }
*/

class Solution {
fun employeeFreeTime(schedule: ArrayList<ArrayList<Interval>>):
ArrayList<Interval> {

}
}
```

Swift:

```
/**
* Definition for an Interval.
* public class Interval {
* public var start: Int
* public var end: Int
* public init(_ start: Int, _ end: Int) {
* self.start = start
* self.end = end
* }
* }
*/

class Solution {
func employeeFreeTime(_ schedule: [[Interval]]) -> [Interval] {

}
}
```

Rust:

```
/*
// Definition for an Interval.
#[derive(PartialEq, Eq, Clone, Debug)]
struct Interval {
pub start:i32,
pub end:i32
}

impl Interval {
#[inline]
fn new(start:i32, end:i32) -> Self{
Interval {
start,
end
}
}
}
*/

impl Solution {
pub fn employee_free_time(schedule: Vec<Vec<Interval>>) -> Vec<Interval> {

}
}
```

Ruby:

```
# Definition for an Interval.
# class Interval
# def initialize(start_, end_)
# @start = start_
# @end = end_
# end
# end

# @param {List[List[Interval]]} schedule
# @return {List[List[Interval]]}
def employeeFreeTime(schedule)

end
```

PHP:

```
/**
* Definition for an Interval.
* class Interval {
* public $start = null;
* public $end = null;
* function __construct($start, $end) {
* $this->start = $start;
* $this->end = $end;
* }
* }
*/

class Solution {
/**
* @param Interval[][] $schedule
* @return Interval[]
*/
function employeeFreeTime($schedule) {

}
}
```

Scala:

```
/**
* Definition for an Interval.
* class Interval(var _start: Int, var _end: Int) {
* var start: Int = _start
* var end: Int = _end
* }
*/

object Solution {
def employeeFreeTime(schedule: List[List[Interval]]): List[Interval] = {

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Employee Free Time
* Difficulty: Hard
* Tags: array, 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
*/

/*
// Definition for an Interval.
class Interval {
public:
int start;
int end;

Interval() {}

Interval(int _start, int _end) {
start = _start;
end = _end;
}
};
*/

class Solution {
public:
vector<Interval> employeeFreeTime(vector<vector<Interval>> schedule) {

}
};
```

Java Solution:

```
/**
* Problem: Employee Free Time
* Difficulty: Hard
* Tags: array, 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
*/

/*
// Definition for an Interval.
class Interval {
public int start;
public int end;

public Interval() {}

public Interval(int _start, int _end) {
start = _start;
end = _end;
}
};
*/

class Solution {
public List<Interval> employeeFreeTime(List<List<Interval>> schedule) {

}
}
```

Python3 Solution:

```
"""
Problem: Employee Free Time
Difficulty: Hard
Tags: array, 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
"""

"""
# Definition for an Interval.
class Interval:
def __init__(self, start: int = None, end: int = None):
self.start = start
self.end = end
"""

class Solution:
def employeeFreeTime(self, schedule: '[[Interval]]') -> '[Interval]':
# TODO: Implement optimized solution
pass
```

Python Solution:

```
"""
# Definition for an Interval.
class Interval(object):
def __init__(self, start=None, end=None):
self.start = start
self.end = end
"""

class Solution(object):
def employeeFreeTime(self, schedule):
"""
:type schedule: [[Interval]]
:rtype: [Interval]
"""
```

JavaScript Solution:

```
/**
* Problem: Employee Free Time
* Difficulty: Hard
* Tags: array, 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
*/

/**
* // Definition for an Interval.
* function Interval(start, end) {
* this.start = start;
* this.end = end;
* };
*/

/**
* @param {Interval[][]} schedule
* @return {Interval[]}
*/
var employeeFreeTime = function(schedule) {

};
```

TypeScript Solution:

```
/**
* Problem: Employee Free Time
* Difficulty: Hard
* Tags: array, 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
*/

/**
* // Definition for an Interval.
* class Interval {
* start: number;
* end: number;
* constructor(start: number, end: number) {
* this.start = start;
* this.end = end;
* }
* }
*/

function employeeFreeTime(schedule: Interval[][]): Interval[] {

};
```

C# Solution:

```
/*
* Problem: Employee Free Time
* Difficulty: Hard
* Tags: array, 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
*/

/*
// Definition for an Interval.
public class Interval {
public int start;
public int end;

public Interval(){}
public Interval(int _start, int _end) {
start = _start;
end = _end;
}
}
*/

public class Solution {
public IList<Interval> EmployeeFreeTime(IList<IList<Interval>> schedule) {

}
}
```

Go Solution:

```
// Problem: Employee Free Time
// Difficulty: Hard
// Tags: array, 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

/**
* Definition for an Interval.
* type Interval struct {
* Start int
* End int
* }
*/

func employeeFreeTime(schedule [][]*Interval) []*Interval {

}
```

Kotlin Solution:

```
/*
* // Definition for an Interval.
* class Interval {
* var start:Int = 0
* var end:Int = 0
*
* constructor(_start:Int, _end:Int) {
* start = _start
* end = _end
* }
* }
*/

class Solution {
fun employeeFreeTime(schedule: ArrayList<ArrayList<Interval>>):
ArrayList<Interval> {

}
}
```

Swift Solution:

```
/**
* Definition for an Interval.
* public class Interval {
* public var start: Int
* public var end: Int
* public init(_ start: Int, _ end: Int) {
* self.start = start
* self.end = end
* }
* }
*/

class Solution {
func employeeFreeTime(_ schedule: [[Interval]]) -> [Interval] {

}
}
```

Rust Solution:

```
// Problem: Employee Free Time
// Difficulty: Hard
// Tags: array, 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

/*
// Definition for an Interval.
#[derive(PartialEq, Eq, Clone, Debug)]
struct Interval {
pub start:i32,
pub end:i32
}

impl Interval {
#[inline]
fn new(start:i32, end:i32) -> Self{
Interval {
start,
end
}
}
}
*/

impl Solution {
```

```
pub fn employee_free_time(schedule: Vec<Vec<Interval>>) -> Vec<Interval> {

}
}
```

Ruby Solution:

```
# Definition for an Interval.
# class Interval
# def initialize(start_, end_)
# @start = start_
# @end = end_
# end
# end

# @param {List[List[Interval]]} schedule
# @return {List[List[Interval]]}
def employeeFreeTime(schedule)

end
```

PHP Solution:

```
/**
* Definition for an Interval.
* class Interval {
* public $start = null;
* public $end = null;
* function __construct($start, $end) {
* $this->start = $start;
* $this->end = $end;
* }
* }
*/

class Solution {
/**
* @param Interval[][] $schedule
* @return Interval[]
*/
function employeeFreeTime($schedule) {



}
}
```

Scala Solution:

```
/**
* Definition for an Interval.
* class Interval(var _start: Int, var _end: Int) {
* var start: Int = _start
* var end: Int = _end
* }
*/

object Solution {
def employeeFreeTime(schedule: List[List[Interval]]): List[Interval] = {

}
}
```