Problem 1942: The Number of the Smallest Unoccupied Chair

Problem Information

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

Problem Description

There is a party where

n

friends numbered from

0

to

n - 1

are attending. There is an

infinite

number of chairs in this party that are numbered from

0

to

infinity

. When a friend arrives at the party, they sit on the unoccupied chair with the smallest number

.

For example, if chairs

0

,

1

, and

5

are occupied when a friend comes, they will sit on chair number

2

.

When a friend leaves the party, their chair becomes unoccupied at the moment they leave. If another friend arrives at that same moment, they can sit in that chair.

You are given a

0-indexed

2D integer array

times

where

times[i] = [arrival

i

, leaving

i

]

, indicating the arrival and leaving times of the

i

th

friend respectively, and an integer

targetFriend

. All arrival times are

distinct

.

Return

the

chair number

that the friend numbered

targetFriend

will sit on

.

Example 1:

Input:

times = [[1,4],[2,3],[4,6]], targetFriend = 1

Output:

1

Explanation:

- Friend 0 arrives at time 1 and sits on chair 0. - Friend 1 arrives at time 2 and sits on chair 1. - Friend 1 leaves at time 3 and chair 1 becomes empty. - Friend 0 leaves at time 4 and chair 0 becomes empty. - Friend 2 arrives at time 4 and sits on chair 0. Since friend 1 sat on chair 1, we return 1.

Example 2:

Input:

times = [[3,10],[1,5],[2,6]], targetFriend = 0

Output:

2

Explanation:

- Friend 1 arrives at time 1 and sits on chair 0. - Friend 2 arrives at time 2 and sits on chair 1. - Friend 0 arrives at time 3 and sits on chair 2. - Friend 1 leaves at time 5 and chair 0 becomes empty. - Friend 2 leaves at time 6 and chair 1 becomes empty. - Friend 0 leaves at time 10 and chair 2 becomes empty. Since friend 0 sat on chair 2, we return 2.

Constraints:

n == times.length

2 <= n <= 10

times[i].length == 2

1 <= arrival

i

< leaving

i

<= 10

5

0 <= targetFriend <= n - 1

Each

arrival

i

time is

distinct

.

Code Snippets

C++:

```
class Solution {
public:
int smallestChair(vector<vector<int>>& times, int targetFriend) {

}
};
```

Java:

```
class Solution {
public int smallestChair(int[][] times, int targetFriend) {

}
}
```

Python3:

```
class Solution:
def smallestChair(self, times: List[List[int]], targetFriend: int) -> int:
```

Python:

```
class Solution(object):
def smallestChair(self, times, targetFriend):
"""
:type times: List[List[int]]
:type targetFriend: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} times
* @param {number} targetFriend
* @return {number}
*/
var smallestChair = function(times, targetFriend) {

};
```

TypeScript:

```
function smallestChair(times: number[][], targetFriend: number): number {

};
```

C#:

```
public class Solution {
public int SmallestChair(int[][] times, int targetFriend) {

}
}
```

C:

```
int smallestChair(int** times, int timesSize, int* timesColSize, int
targetFriend) {

}
```

Go:

```
func smallestChair(times [][]int, targetFriend int) int {

}
```

Kotlin:

```
class Solution {
fun smallestChair(times: Array<IntArray>, targetFriend: Int): Int {

}
}
```

Swift:

```
class Solution {
func smallestChair(_ times: [[Int]], _ targetFriend: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn smallest_chair(times: Vec<Vec<i32>>, target_friend: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} times
# @param {Integer} target_friend
# @return {Integer}
def smallest_chair(times, target_friend)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $times
* @param Integer $targetFriend
* @return Integer
*/
function smallestChair($times, $targetFriend) {

}
}
```

Dart:

```
class Solution {
int smallestChair(List<List<int>> times, int targetFriend) {

}
}
```

Scala:

```
object Solution {
def smallestChair(times: Array[Array[Int]], targetFriend: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec smallest_chair(times :: [[integer]], target_friend :: integer) ::
```

```
integer
def smallest_chair(times, target_friend) do

end
end
```

Erlang:

```
-spec smallest_chair(Times :: [[integer()]], TargetFriend :: integer()) ->
integer().
smallest_chair(Times, TargetFriend) ->
.
```

Racket:

```
(define/contract (smallest-chair times targetFriend)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: The Number of the Smallest Unoccupied Chair
* Difficulty: Medium
* Tags: array, hash, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
int smallestChair(vector<vector<int>>& times, int targetFriend) {

}
};
```

Java Solution:

```
/**
* Problem: The Number of the Smallest Unoccupied Chair
* Difficulty: Medium
* Tags: array, hash, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int smallestChair(int[][] times, int targetFriend) {

}
}
```

Python3 Solution:

```
"""
Problem: The Number of the Smallest Unoccupied Chair
Difficulty: Medium
Tags: array, hash, queue, heap

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def smallestChair(self, times: List[List[int]], targetFriend: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def smallestChair(self, times, targetFriend):
"""
:type times: List[List[int]]
:type targetFriend: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: The Number of the Smallest Unoccupied Chair
* Difficulty: Medium
* Tags: array, hash, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number[][]} times
* @param {number} targetFriend
* @return {number}
*/
var smallestChair = function(times, targetFriend) {

};
```

TypeScript Solution:

```
/**
* Problem: The Number of the Smallest Unoccupied Chair
* Difficulty: Medium
* Tags: array, hash, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function smallestChair(times: number[][], targetFriend: number): number {

};
```

C# Solution:

```
/*
* Problem: The Number of the Smallest Unoccupied Chair
* Difficulty: Medium
* Tags: array, hash, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int SmallestChair(int[][] times, int targetFriend) {

}
}
```

C Solution:

```
/*
* Problem: The Number of the Smallest Unoccupied Chair
* Difficulty: Medium
* Tags: array, hash, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int smallestChair(int** times, int timesSize, int* timesColSize, int
targetFriend) {

}
```

Go Solution:

```
// Problem: The Number of the Smallest Unoccupied Chair
// Difficulty: Medium
// Tags: array, hash, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func smallestChair(times [][]int, targetFriend int) int {
}
```

Kotlin Solution:

```
class Solution {
fun smallestChair(times: Array<IntArray>, targetFriend: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func smallestChair(_ times: [[Int]], _ targetFriend: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: The Number of the Smallest Unoccupied Chair
// Difficulty: Medium
// Tags: array, hash, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn smallest_chair(times: Vec<Vec<i32>>, target_friend: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} times
# @param {Integer} target_friend
# @return {Integer}
def smallest_chair(times, target_friend)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $times
* @param Integer $targetFriend
* @return Integer
*/
function smallestChair($times, $targetFriend) {

}
}
```

Dart Solution:

```
class Solution {
int smallestChair(List<List<int>> times, int targetFriend) {

}
}
```

Scala Solution:

```
object Solution {
def smallestChair(times: Array[Array[Int]], targetFriend: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec smallest_chair(times :: [[integer]], target_friend :: integer) ::
integer
def smallest_chair(times, target_friend) do

end
```

```
end
```

Erlang Solution:

```
-spec smallest_chair(Times :: [[integer()]], TargetFriend :: integer()) ->
integer().
smallest_chair(Times, TargetFriend) ->
.
```

Racket Solution:

```
(define/contract (smallest-chair times targetFriend)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```