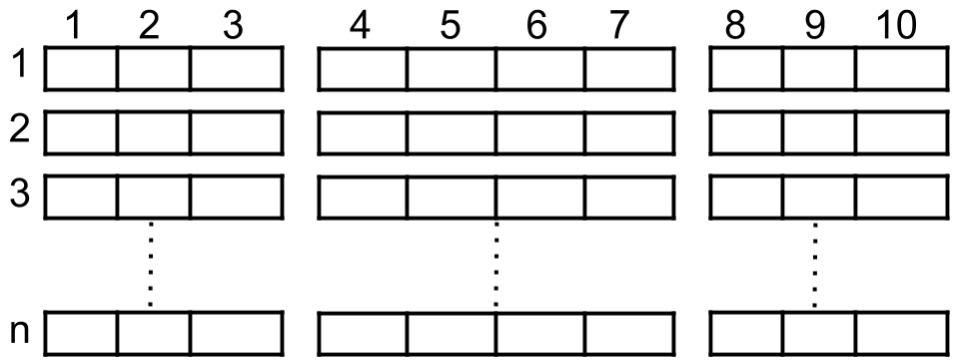
Problem 1386: Cinema Seat Allocation

Problem Information

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

Problem Description



A cinema has

n

rows of seats, numbered from 1 to

n

and there are ten seats in each row, labelled from 1 to 10 as shown in the figure above.

Given the array

reservedSeats

containing the numbers of seats already reserved, for example,

reservedSeats[i] = [3,8]

means the seat located in row

3

and labelled with

8

is already reserved.

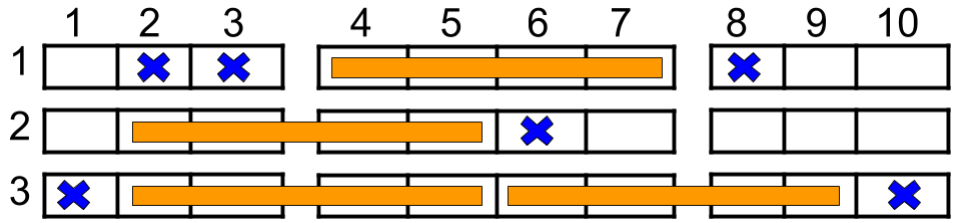
Return the maximum number of four-person groups you can assign on the cinema seats.

A four-person group occupies four adjacent seats

in one single row

. Seats across an aisle (such as [3,3] and [3,4]) are not considered to be adjacent, but there is an exceptional case on which an aisle split a four-person group, in that case, the aisle split a four-person group in the middle, which means to have two people on each side.

Example 1:



Input:

n = 3, reservedSeats = [[1,2],[1,3],[1,8],[2,6],[3,1],[3,10]]

Output:

4

Explanation:

The figure above shows the optimal allocation for four groups, where seats mark with blue are already reserved and contiguous seats mark with orange are for one group.

Example 2:

Input:

n = 2, reservedSeats = [[2,1],[1,8],[2,6]]

Output:

2

Example 3:

Input:

n = 4, reservedSeats = [[4,3],[1,4],[4,6],[1,7]]

Output:

4

Constraints:

1 <= n <= 10^9

1 <= reservedSeats.length <= min(10*n, 10^4)

reservedSeats[i].length == 2

1 <= reservedSeats[i][0] <= n

1 <= reservedSeats[i][1] <= 10

All

reservedSeats[i]

are distinct.

Code Snippets

C++:

```
class Solution {
public:
int maxNumberOfFamilies(int n, vector<vector<int>>& reservedSeats) {

}
};
```

Java:

```
class Solution {
public int maxNumberOfFamilies(int n, int[][] reservedSeats) {

}
}
```

Python3:

```
class Solution:
def maxNumberOfFamilies(self, n: int, reservedSeats: List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def maxNumberOfFamilies(self, n, reservedSeats):
"""
:type n: int
:type reservedSeats: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} reservedSeats
```

```
* @return {number}
*/
var maxNumberOfFamilies = function(n, reservedSeats) {

};
```

TypeScript:

```
function maxNumberOfFamilies(n: number, reservedSeats: number[][]): number {

};
```

C#:

```
public class Solution {
public int MaxNumberOfFamilies(int n, int[][] reservedSeats) {

}
}
```

C:

```
int maxNumberOfFamilies(int n, int** reservedSeats, int reservedSeatsSize,
int* reservedSeatsColSize) {

}
```

Go:

```
func maxNumberOfFamilies(n int, reservedSeats [][]int) int {

}
```

Kotlin:

```
class Solution {
fun maxNumberOfFamilies(n: Int, reservedSeats: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func maxNumberOfFamilies(_ n: Int, _ reservedSeats: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_number_of_families(n: i32, reserved_seats: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} reserved_seats
# @return {Integer}
def max_number_of_families(n, reserved_seats)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $reservedSeats
* @return Integer
*/
function maxNumberOfFamilies($n, $reservedSeats) {

}
}
```

Dart:

```
class Solution {
int maxNumberOfFamilies(int n, List<List<int>> reservedSeats) {

}
```

```
}
```

Scala:

```
object Solution {
def maxNumberOfFamilies(n: Int, reservedSeats: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_number_of_families(n :: integer, reserved_seats :: [[integer]]) ::
integer
def max_number_of_families(n, reserved_seats) do

end
end
```

Erlang:

```
-spec max_number_of_families(N :: integer(), ReservedSeats :: [[integer()]])
-> integer().
max_number_of_families(N, ReservedSeats) ->
.
```

Racket:

```
(define/contract (max-number-of-families n reservedSeats)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Cinema Seat Allocation
* Difficulty: Medium
```

```
* Tags: array, greedy, hash
*
* 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 maxNumberOfFamilies(int n, vector<vector<int>>& reservedSeats) {

}
};
```

Java Solution:

```
/**
* Problem: Cinema Seat Allocation
* Difficulty: Medium
* Tags: array, greedy, hash
*
* 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 maxNumberOfFamilies(int n, int[][] reservedSeats) {

}
}
```

Python3 Solution:

```
"""
Problem: Cinema Seat Allocation
Difficulty: Medium
Tags: array, greedy, hash

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 maxNumberOfFamilies(self, n: int, reservedSeats: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxNumberOfFamilies(self, n, reservedSeats):
"""
:type n: int
:type reservedSeats: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Cinema Seat Allocation
* Difficulty: Medium
* Tags: array, greedy, hash
*
* 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} n
* @param {number[][]} reservedSeats
* @return {number}
*/
var maxNumberOfFamilies = function(n, reservedSeats) {

};
```

TypeScript Solution:

```
/**
* Problem: Cinema Seat Allocation
* Difficulty: Medium
* Tags: array, greedy, hash
*
* 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 maxNumberOfFamilies(n: number, reservedSeats: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Cinema Seat Allocation
* Difficulty: Medium
* Tags: array, greedy, hash
*
* 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 MaxNumberOfFamilies(int n, int[][] reservedSeats) {

}
}
```

C Solution:

```
/*
* Problem: Cinema Seat Allocation
* Difficulty: Medium
* Tags: array, greedy, hash
*
* 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 maxNumberOfFamilies(int n, int** reservedSeats, int reservedSeatsSize,
int* reservedSeatsColSize) {

}
```

Go Solution:

```
// Problem: Cinema Seat Allocation
// Difficulty: Medium
// Tags: array, greedy, hash
//
// 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 maxNumberOfFamilies(n int, reservedSeats [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxNumberOfFamilies(n: Int, reservedSeats: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxNumberOfFamilies(_ n: Int, _ reservedSeats: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Cinema Seat Allocation
// Difficulty: Medium
```

```
// Tags: array, greedy, hash
//
// 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 max_number_of_families(n: i32, reserved_seats: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} reserved_seats
# @return {Integer}
def max_number_of_families(n, reserved_seats)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $reservedSeats
* @return Integer
*/
function maxNumberOfFamilies($n, $reservedSeats) {

}
}
```

Dart Solution:

```
class Solution {
int maxNumberOfFamilies(int n, List<List<int>> reservedSeats) {

}
```

```
}
```

Scala Solution:

```
object Solution {
def maxNumberOfFamilies(n: Int, reservedSeats: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_number_of_families(n :: integer, reserved_seats :: [[integer]]) ::
integer
def max_number_of_families(n, reserved_seats) do

end
end
```

Erlang Solution:

```
-spec max_number_of_families(N :: integer(), ReservedSeats :: [[integer()]])
-> integer().
max_number_of_families(N, ReservedSeats) ->
.
```

Racket Solution:

```
(define/contract (max-number-of-families n reservedSeats)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```