Problem 1847: Closest Room

Problem Information

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

Problem Description

There is a hotel with

n

rooms. The rooms are represented by a 2D integer array

rooms

where

rooms[i] = [roomId

i

, size

i

]

denotes that there is a room with room number

roomId

i

and size equal to

size

i

. Each

roomId

i

is guaranteed to be

unique

.

You are also given

k

queries in a 2D array

queries

where

queries[j] = [preferred

j

, minSize

j

]

. The answer to the

j

th

query is the room number

id

of a room such that:

The room has a size of

at least

minSize

j

, and

abs(id - preferred

j

)

is

minimized

, where

abs(x)

is the absolute value of

x

.

If there is a

tie

in the absolute difference, then use the room with the

smallest

such

id

. If there is

no such room

, the answer is

-1

.

Return

an array

answer

of length

k

where

answer[j]

contains the answer to the

j

th

query

.

Example 1:

Input:

rooms = [[2,2],[1,2],[3,2]], queries = [[3,1],[3,3],[5,2]]

Output:

[3,-1,3]

Explanation:

The answers to the queries are as follows: Query = [3,1]: Room number 3 is the closest as abs(3 - 3) = 0, and its size of 2 is at least 1. The answer is 3. Query = [3,3]: There are no rooms with a size of at least 3, so the answer is -1. Query = [5,2]: Room number 3 is the closest as abs(3 - 5) = 2, and its size of 2 is at least 2. The answer is 3.

Example 2:

Input:

rooms = [[1,4],[2,3],[3,5],[4,1],[5,2]], queries = [[2,3],[2,4],[2,5]]

Output:

[2,1,3]

Explanation:

The answers to the queries are as follows: Query = [2,3]: Room number 2 is the closest as abs(2 - 2) = 0, and its size of 3 is at least 3. The answer is 2. Query = [2,4]: Room numbers 1 and 3 both have sizes of at least 4. The answer is 1 since it is smaller. Query = [2,5]: Room number 3 is the only room with a size of at least 5. The answer is 3.

Constraints:

n == rooms.length

1 <= n <= 10

5

k == queries.length

1 <= k <= 10

4

1 <= roomId

i

, preferred

j

<= 10

7

1 <= size

i

, minSize

j

<= 10

Code Snippets

C++:

```
class Solution {
public:
vector<int> closestRoom(vector<vector<int>>& rooms, vector<vector<int>>&
queries) {

}
};
```

Java:

```
class Solution {
public int[] closestRoom(int[][] rooms, int[][] queries) {

}
}
```

Python3:

```
class Solution:
def closestRoom(self, rooms: List[List[int]], queries: List[List[int]]) ->
List[int]:
```

Python:

```
class Solution(object):
def closestRoom(self, rooms, queries):
"""
:type rooms: List[List[int]]
:type queries: List[List[int]]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[][]} rooms
* @param {number[][]} queries
* @return {number[]}
*/
var closestRoom = function(rooms, queries) {

};
```

TypeScript:

```
function closestRoom(rooms: number[][], queries: number[][]): number[] {

};
```

C#:

```
public class Solution {
public int[] ClosestRoom(int[][] rooms, int[][] queries) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* closestRoom(int** rooms, int roomsSize, int* roomsColSize, int**
queries, int queriesSize, int* queriesColSize, int* returnSize) {

}
```

Go:

```
func closestRoom(rooms [][]int, queries [][]int) []int {

}
```

Kotlin:

```
class Solution {
fun closestRoom(rooms: Array<IntArray>, queries: Array<IntArray>): IntArray {
}
}
```

Swift:

```
class Solution {
func closestRoom(_ rooms: [[Int]], _ queries: [[Int]]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn closest_room(rooms: Vec<Vec<i32>>, queries: Vec<Vec<i32>>) -> Vec<i32>
{

}
}
```

Ruby:

```
# @param {Integer[][]} rooms
# @param {Integer[][]} queries
# @return {Integer[]}
def closest_room(rooms, queries)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $rooms
* @param Integer[][] $queries
* @return Integer[]
*/
function closestRoom($rooms, $queries) {

}
}
```

Dart:

```
class Solution {
List<int> closestRoom(List<List<int>> rooms, List<List<int>> queries) {

}
}
```

Scala:

```
object Solution {
def closestRoom(rooms: Array[Array[Int]], queries: Array[Array[Int]]):
Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec closest_room(rooms :: [[integer]], queries :: [[integer]]) :: [integer]
def closest_room(rooms, queries) do

end
end
```

Erlang:

```
-spec closest_room(Rooms :: [[integer()]], Queries :: [[integer()]]) ->
[integer()].
closest_room(Rooms, Queries) ->
.
```

Racket:

```
(define/contract (closest-room rooms queries)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)) (listof
exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Closest Room
* Difficulty: Hard
* Tags: array, sort, search
*
* 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:
vector<int> closestRoom(vector<vector<int>>& rooms, vector<vector<int>>& queries) {

}
};
```

Java Solution:

```
/**
* Problem: Closest Room
* Difficulty: Hard
* Tags: array, sort, search
*
* 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[] closestRoom(int[][] rooms, int[][] queries) {

}
}
```

Python3 Solution:

```
"""
Problem: Closest Room
Difficulty: Hard
Tags: array, sort, search

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 closestRoom(self, rooms: List[List[int]], queries: List[List[int]]) ->
List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def closestRoom(self, rooms, queries):
"""
:type rooms: List[List[int]]
:type queries: List[List[int]]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Closest Room
* Difficulty: Hard
* Tags: array, sort, search
*
* 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[][]} rooms
* @param {number[][]} queries
* @return {number[]}
```

```
*/
var closestRoom = function(rooms, queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Closest Room
* Difficulty: Hard
* Tags: array, sort, search
*
* 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 closestRoom(rooms: number[][], queries: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Closest Room
* Difficulty: Hard
* Tags: array, sort, search
*
* 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[] ClosestRoom(int[][] rooms, int[][] queries) {

}
}
```

C Solution:

```
/*
* Problem: Closest Room
* Difficulty: Hard
* Tags: array, sort, search
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* closestRoom(int** rooms, int roomsSize, int* roomsColSize, int**
queries, int queriesSize, int* queriesColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Closest Room
// Difficulty: Hard
// Tags: array, sort, search
//
// 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 closestRoom(rooms [][]int, queries [][]int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun closestRoom(rooms: Array<IntArray>, queries: Array<IntArray>): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func closestRoom(_ rooms: [[Int]], _ queries: [[Int]]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Closest Room
// Difficulty: Hard
// Tags: array, sort, search
//
// 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 closest_room(rooms: Vec<Vec<i32>>, queries: Vec<Vec<i32>>) -> Vec<i32>
{

}
}
```

Ruby Solution:

```
# @param {Integer[][]} rooms
# @param {Integer[][]} queries
# @return {Integer[]}
def closest_room(rooms, queries)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $rooms
* @param Integer[][] $queries
* @return Integer[]
*/
function closestRoom($rooms, $queries) {
}
}
```

Dart Solution:

```
class Solution {
List<int> closestRoom(List<List<int>> rooms, List<List<int>> queries) {

}
}
```

Scala Solution:

```
object Solution {
def closestRoom(rooms: Array[Array[Int]], queries: Array[Array[Int]]):
Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec closest_room(rooms :: [[integer]], queries :: [[integer]]) :: [integer]
def closest_room(rooms, queries) do

end
end
```

Erlang Solution:

```
-spec closest_room(Rooms :: [[integer()]], Queries :: [[integer()]]) ->
[integer()].
closest_room(Rooms, Queries) ->
.
```

Racket Solution:

```
(define/contract (closest-room rooms queries)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)) (listof
exact-integer?))
)
```