Problem 2271: Maximum White Tiles Covered by a Carpet

Problem Information

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

Problem Description

You are given a 2D integer array

tiles

where

tiles[i] = [l

i

, r

i

]

represents that every tile

j

in the range

l

i

<= j <= r

i

is colored white.

You are also given an integer

carpetLen

, the length of a single carpet that can be placed

anywhere

.

Return

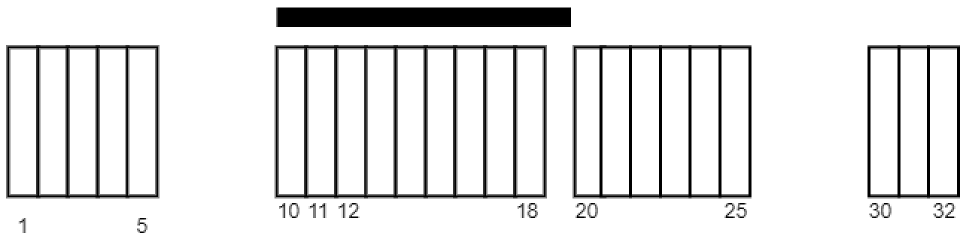
the

maximum

number of white tiles that can be covered by the carpet

.

Example 1:



Input:

tiles = [[1,5],[10,11],[12,18],[20,25],[30,32]], carpetLen = 10

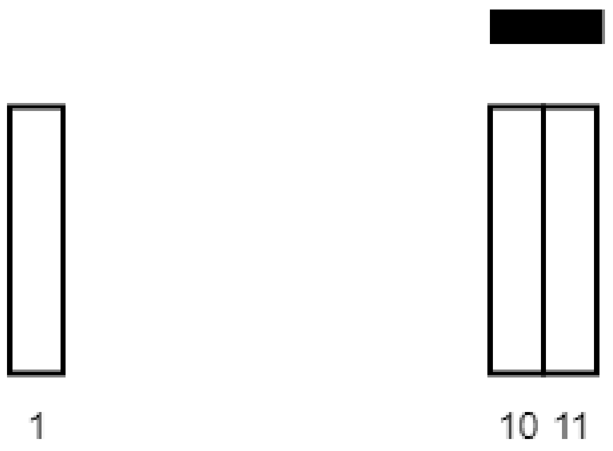
Output:

9

Explanation:

Place the carpet starting on tile 10. It covers 9 white tiles, so we return 9. Note that there may be other places where the carpet covers 9 white tiles. It can be shown that the carpet cannot cover more than 9 white tiles.

Example 2:



Input:

tiles = [[10,11],[1,1]], carpetLen = 2

Output:

2

Explanation:

Place the carpet starting on tile 10. It covers 2 white tiles, so we return 2.

Constraints:

1 <= tiles.length <= 5 * 10

4

tiles[i].length == 2

1 <= l

i

<= r

i

<= 10

9

1 <= carpetLen <= 10

9

The

tiles

are

non-overlapping

.

Code Snippets

C++:

```
class Solution {
public:
int maximumWhiteTiles(vector<vector<int>>& tiles, int carpetLen) {

}
};
```

Java:

```
class Solution {
public int maximumWhiteTiles(int[][] tiles, int carpetLen) {

}
}
```

Python3:

```
class Solution:
def maximumWhiteTiles(self, tiles: List[List[int]], carpetLen: int) -> int:
```

Python:

```
class Solution(object):
def maximumWhiteTiles(self, tiles, carpetLen):
"""
:type tiles: List[List[int]]
:type carpetLen: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} tiles
* @param {number} carpetLen
* @return {number}
*/
var maximumWhiteTiles = function(tiles, carpetLen) {

};
```

TypeScript:

```
function maximumWhiteTiles(tiles: number[][], carpetLen: number): number {

};
```

C#:

}

```
public class Solution {
public int MaximumWhiteTiles(int[][] tiles, int carpetLen) {

}
}
```

C:

```
int maximumWhiteTiles(int** tiles, int tilesSize, int* tilesColSize, int
carpetLen) {

}
```

Go:

```
func maximumWhiteTiles(tiles [][]int, carpetLen int) int {

}
```

Kotlin:

```
class Solution {
fun maximumWhiteTiles(tiles: Array<IntArray>, carpetLen: Int): Int {

}
}
```

Swift:

```
class Solution {
func maximumWhiteTiles(_ tiles: [[Int]], _ carpetLen: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn maximum_white_tiles(tiles: Vec<Vec<i32>>, carpet_len: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} tiles
# @param {Integer} carpet_len
# @return {Integer}
def maximum_white_tiles(tiles, carpet_len)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $tiles
* @param Integer $carpetLen
* @return Integer
*/
function maximumWhiteTiles($tiles, $carpetLen) {

}
}
```

Dart:

```
class Solution {
int maximumWhiteTiles(List<List<int>> tiles, int carpetLen) {

}
}
```

Scala:

```
object Solution {
def maximumWhiteTiles(tiles: Array[Array[Int]], carpetLen: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec maximum_white_tiles(tiles :: [[integer]], carpet_len :: integer) ::
```

```
integer
def maximum_white_tiles(tiles, carpet_len) do

end
end
```

Erlang:

```
-spec maximum_white_tiles(Tiles :: [[integer()]], CarpetLen :: integer()) ->
integer().
maximum_white_tiles(Tiles, CarpetLen) ->
.
```

Racket:

```
(define/contract (maximum-white-tiles tiles carpetLen)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum White Tiles Covered by a Carpet
* Difficulty: Medium
* Tags: array, greedy, 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 maximumWhiteTiles(vector<vector<int>>& tiles, int carpetLen) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum White Tiles Covered by a Carpet
* Difficulty: Medium
* Tags: array, greedy, 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 maximumWhiteTiles(int[][] tiles, int carpetLen) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum White Tiles Covered by a Carpet
Difficulty: Medium
Tags: array, greedy, 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 maximumWhiteTiles(self, tiles: List[List[int]], carpetLen: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maximumWhiteTiles(self, tiles, carpetLen):
"""
:type tiles: List[List[int]]
:type carpetLen: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum White Tiles Covered by a Carpet
* Difficulty: Medium
* Tags: array, greedy, 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[][]} tiles
* @param {number} carpetLen
* @return {number}
*/
var maximumWhiteTiles = function(tiles, carpetLen) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum White Tiles Covered by a Carpet
* Difficulty: Medium
* Tags: array, greedy, 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 maximumWhiteTiles(tiles: number[][], carpetLen: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximum White Tiles Covered by a Carpet
* Difficulty: Medium
* Tags: array, greedy, 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 MaximumWhiteTiles(int[][] tiles, int carpetLen) {

}
}
```

C Solution:

```
/*
* Problem: Maximum White Tiles Covered by a Carpet
* Difficulty: Medium
* Tags: array, greedy, 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
*/

int maximumWhiteTiles(int** tiles, int tilesSize, int* tilesColSize, int
carpetLen) {

}
```

Go Solution:

```
// Problem: Maximum White Tiles Covered by a Carpet
// Difficulty: Medium
// Tags: array, greedy, 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 maximumWhiteTiles(tiles [][]int, carpetLen int) int {
}
```

Kotlin Solution:

```
class Solution {
fun maximumWhiteTiles(tiles: Array<IntArray>, carpetLen: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func maximumWhiteTiles(_ tiles: [[Int]], _ carpetLen: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum White Tiles Covered by a Carpet
// Difficulty: Medium
// Tags: array, greedy, 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 maximum_white_tiles(tiles: Vec<Vec<i32>>, carpet_len: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} tiles
# @param {Integer} carpet_len
# @return {Integer}
def maximum_white_tiles(tiles, carpet_len)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $tiles
* @param Integer $carpetLen
* @return Integer
*/
function maximumWhiteTiles($tiles, $carpetLen) {

}
}
```

Dart Solution:

```
class Solution {
int maximumWhiteTiles(List<List<int>> tiles, int carpetLen) {

}
}
```

Scala Solution:

```
object Solution {
def maximumWhiteTiles(tiles: Array[Array[Int]], carpetLen: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximum_white_tiles(tiles :: [[integer]], carpet_len :: integer) ::
integer
def maximum_white_tiles(tiles, carpet_len) do

end
```

```
end
```

Erlang Solution:

```
-spec maximum_white_tiles(Tiles :: [[integer()]], CarpetLen :: integer()) ->
integer().
maximum_white_tiles(Tiles, CarpetLen) ->
.
```

Racket Solution:

```
(define/contract (maximum-white-tiles tiles carpetLen)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```