Problem 2158: Amount of New Area Painted Each Day

Problem Information

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

Problem Description

There is a long and thin painting that can be represented by a number line. You are given a

0-indexed

2D integer array

paint

of length

n

, where

paint[i] = [start

i

, end

i

]

. This means that on the

i

th

day you need to paint the area

between

start

i

and

end

i

.

Painting the same area multiple times will create an uneven painting so you only want to paint each area of the painting at most

once

.

Return

an integer array

worklog

of length

n

, where

worklog[i]

is the amount of

new

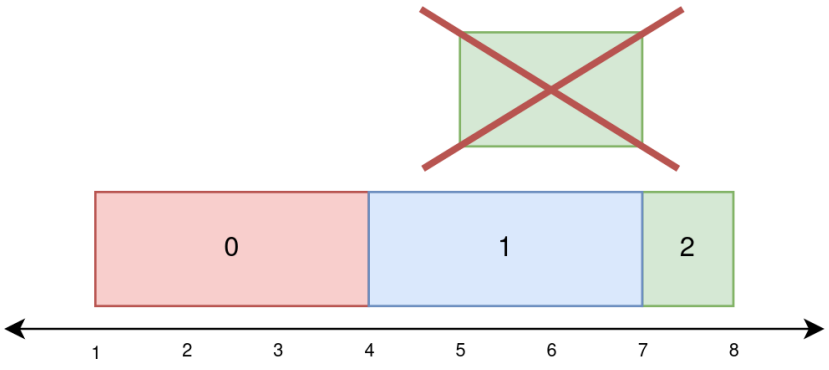
area that you painted on the

i

th

day.

Example 1:



Input:

paint = [[1,4],[4,7],[5,8]]

Output:

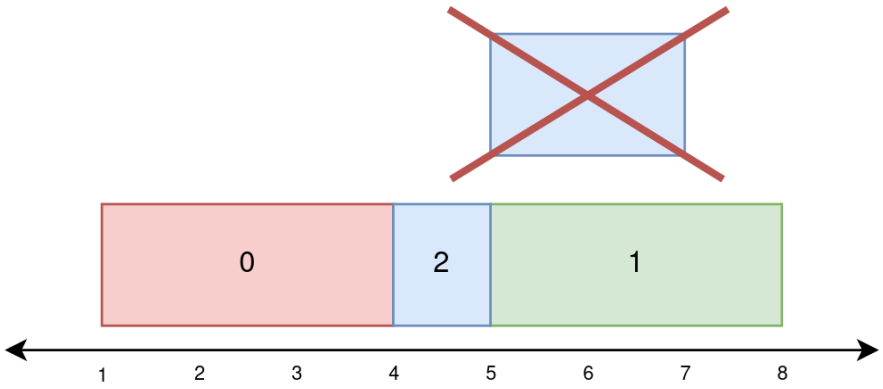
[3,3,1]

Explanation:

On day 0, paint everything between 1 and 4. The amount of new area painted on day 0 is 4 - 1 = 3. On day 1, paint everything between 4 and 7. The amount of new area painted on day 1 is

7 - 4 = 3. On day 2, paint everything between 7 and 8. Everything between 5 and 7 was already painted on day 1. The amount of new area painted on day 2 is 8 - 7 = 1.

Example 2:



Input:

paint = [[1,4],[5,8],[4,7]]

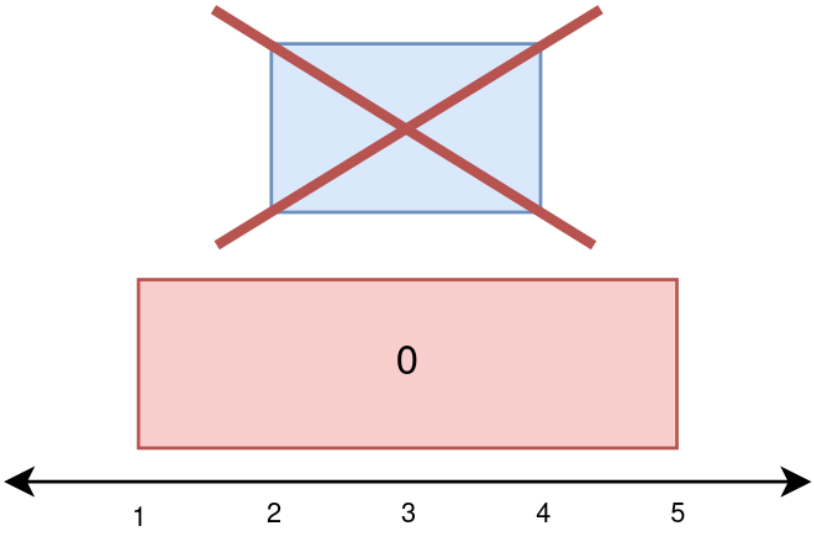
Output:

[3,3,1]

Explanation:

On day 0, paint everything between 1 and 4. The amount of new area painted on day 0 is 4 - 1 = 3. On day 1, paint everything between 5 and 8. The amount of new area painted on day 1 is 8 - 5 = 3. On day 2, paint everything between 4 and 5. Everything between 5 and 7 was already painted on day 1. The amount of new area painted on day 2 is 5 - 4 = 1.

Example 3:



Input:

paint = [[1,5],[2,4]]

Output:

[4,0]

Explanation:

On day 0, paint everything between 1 and 5. The amount of new area painted on day 0 is 5 - 1 = 4. On day 1, paint nothing because everything between 2 and 4 was already painted on day 0. The amount of new area painted on day 1 is 0.

Constraints:

1 <= paint.length <= 10

5

paint[i].length == 2

0 <= start

i

< end

i

<= 5 * 10

4

Code Snippets

C++:

```
class Solution {
public:
vector<int> amountPainted(vector<vector<int>>& paint) {

}
};
```

Java:

```
class Solution {
public int[] amountPainted(int[][] paint) {

}
}
```

Python3:

```
class Solution:
def amountPainted(self, paint: List[List[int]]) -> List[int]:
```

Python:

```
class Solution(object):
def amountPainted(self, paint):
"""
:type paint: List[List[int]]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[][]} paint
* @return {number[]}
*/
var amountPainted = function(paint) {

};
```

TypeScript:

```
function amountPainted(paint: number[][]): number[] {

};
```

C#:

```
public class Solution {
public int[] AmountPainted(int[][] paint) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* amountPainted(int** paint, int paintSize, int* paintColSize, int*
returnSize) {

}
```

Go:

```
func amountPainted(paint [][]int) []int {

}
```

Kotlin:

```
class Solution {
fun amountPainted(paint: Array<IntArray>): IntArray {

}
}
```

Swift:

```
class Solution {
func amountPainted(_ paint: [[Int]]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn amount_painted(paint: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[][]} paint
# @return {Integer[]}
def amount_painted(paint)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $paint
* @return Integer[]
*/
function amountPainted($paint) {

}
}
```

Dart:

```
class Solution {
List<int> amountPainted(List<List<int>> paint) {

}
}
```

Scala:

```
object Solution {
def amountPainted(paint: Array[Array[Int]]): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec amount_painted(paint :: [[integer]]) :: [integer]
def amount_painted(paint) do

end
end
```

Erlang:

```
-spec amount_painted(Paint :: [[integer()]]) -> [integer()].
amount_painted(Paint) ->
.
```

Racket:

```
(define/contract (amount-painted paint)
(-> (listof (listof exact-integer?)) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Amount of New Area Painted Each Day
* Difficulty: Hard
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
vector<int> amountPainted(vector<vector<int>>& paint) {

}
};
```

Java Solution:

```
/**
* Problem: Amount of New Area Painted Each Day
* Difficulty: Hard
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public int[] amountPainted(int[][] paint) {

}
}
```

Python3 Solution:

```
"""
Problem: Amount of New Area Painted Each Day
Difficulty: Hard
Tags: array, tree
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

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

Python Solution:

```
class Solution(object):
def amountPainted(self, paint):
"""
:type paint: List[List[int]]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Amount of New Area Painted Each Day
* Difficulty: Hard
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {number[][]} paint
* @return {number[]}
*/
var amountPainted = function(paint) {

};
```

TypeScript Solution:

```
/**
* Problem: Amount of New Area Painted Each Day
* Difficulty: Hard
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function amountPainted(paint: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Amount of New Area Painted Each Day
* Difficulty: Hard
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public int[] AmountPainted(int[][] paint) {

}
}
```

C Solution:

```
/*
* Problem: Amount of New Area Painted Each Day
* Difficulty: Hard
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
```

```
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* amountPainted(int** paint, int paintSize, int* paintColSize, int*
returnSize) {

}
```

Go Solution:

```
// Problem: Amount of New Area Painted Each Day
// Difficulty: Hard
// Tags: array, tree
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func amountPainted(paint [][]int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun amountPainted(paint: Array<IntArray>): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func amountPainted(_ paint: [[Int]]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Amount of New Area Painted Each Day
// Difficulty: Hard
// Tags: array, tree
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn amount_painted(paint: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} paint
# @return {Integer[]}
def amount_painted(paint)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $paint
* @return Integer[]
*/
function amountPainted($paint) {

}
}
```

Dart Solution:

```
class Solution {
List<int> amountPainted(List<List<int>> paint) {

}
}
```

Scala Solution:

```
object Solution {
def amountPainted(paint: Array[Array[Int]]): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec amount_painted(paint :: [[integer]]) :: [integer]
def amount_painted(paint) do

end
end
```

Erlang Solution:

```
-spec amount_painted(Paint :: [[integer()]]) -> [integer()].
amount_painted(Paint) ->
.
```

Racket Solution:

```
(define/contract (amount-painted paint)
(-> (listof (listof exact-integer?)) (listof exact-integer?))
)
```