Problem 1943: Describe the Painting

Problem Information

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

Problem Description

There is a long and thin painting that can be represented by a number line. The painting was painted with multiple overlapping segments where each segment was painted with a

unique

color. You are given a 2D integer array

segments

, where

segments[i] = [start

i

, end

i

, color

i

]

represents the

half-closed segment

[start

i

, end

i

)

with

color

i

as the color.

The colors in the overlapping segments of the painting were

mixed

when it was painted. When two or more colors mix, they form a new color that can be represented as a

set

of mixed colors.

For example, if colors

2

,

4

, and

6

are mixed, then the resulting mixed color is

{2,4,6}

.

For the sake of simplicity, you should only output the

sum

of the elements in the set rather than the full set.

You want to

describe

the painting with the

minimum

number of non-overlapping

half-closed segments

of these mixed colors. These segments can be represented by the 2D array

painting

where

painting[j] = [left

j

, right

j

, mix

j

]

describes a

half-closed segment

[left

j

, right

j

)

with the mixed color

sum

of

mix

j

.

For example, the painting created with

segments = [[1,4,5],[1,7,7]]

can be described by

painting = [[1,4,12],[4,7,7]]

because:

[1,4)

is colored

{5,7}

(with a sum of

12

) from both the first and second segments.

[4,7)

is colored

{7}

from only the second segment.

Return

the 2D array

painting

describing the finished painting (excluding any parts that are

not

painted). You may return the segments in

any order

.

A

half-closed segment

[a, b)

is the section of the number line between points

a

and

b

including

point

a

and

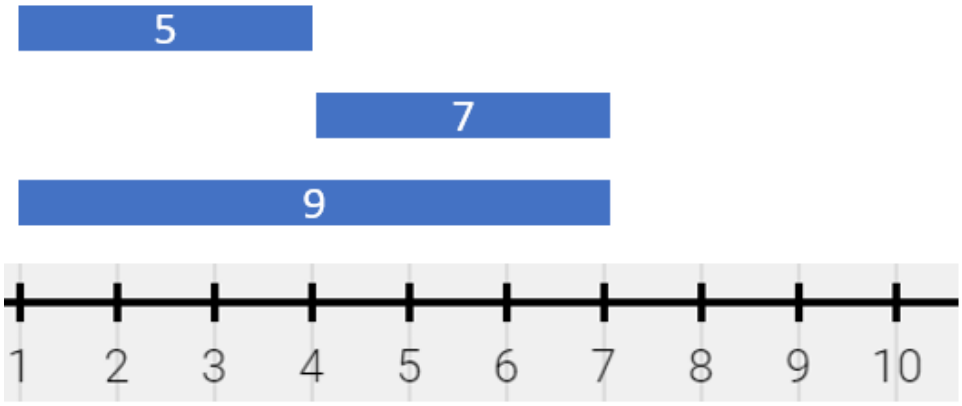
not including

point

b

.

Example 1:



Input:

segments = [[1,4,5],[4,7,7],[1,7,9]]

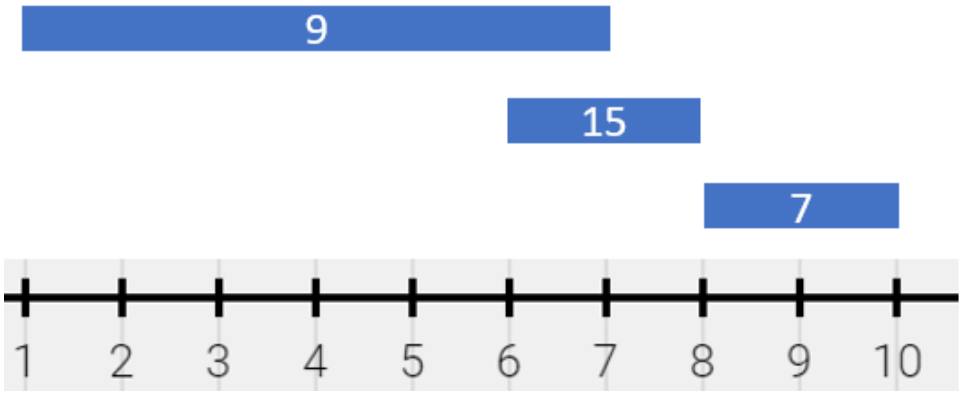
Output:

[[1,4,14],[4,7,16]]

Explanation:

The painting can be described as follows: - [1,4) is colored {5,9} (with a sum of 14) from the first and third segments. - [4,7) is colored {7,9} (with a sum of 16) from the second and third segments.

Example 2:



Input:

segments = [[1,7,9],[6,8,15],[8,10,7]]

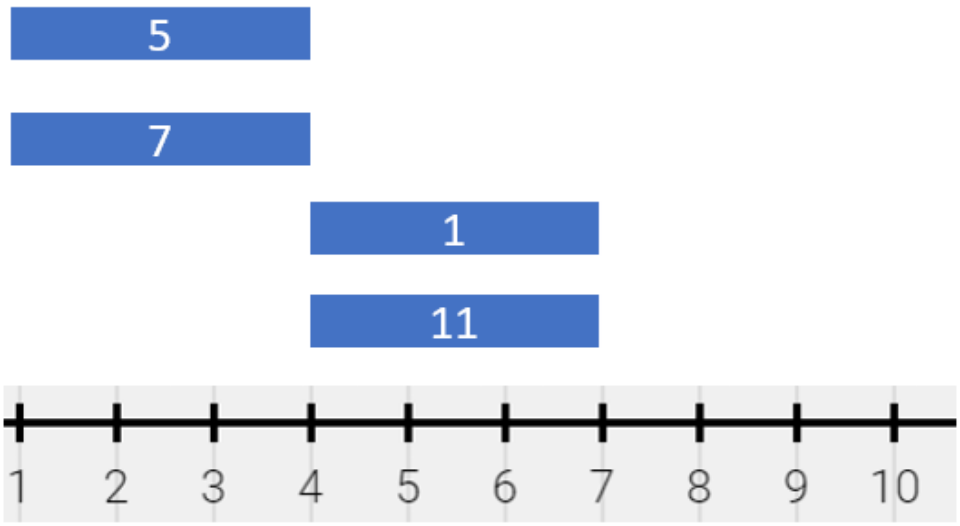
Output:

[[1,6,9],[6,7,24],[7,8,15],[8,10,7]]

Explanation:

The painting can be described as follows: - [1,6) is colored 9 from the first segment. - [6,7) is colored {9,15} (with a sum of 24) from the first and second segments. - [7,8) is colored 15 from the second segment. - [8,10) is colored 7 from the third segment.

Example 3:



Input:

segments = [[1,4,5],[1,4,7],[4,7,1],[4,7,11]]

Output:

[[1,4,12],[4,7,12]]

Explanation:

The painting can be described as follows: - [1,4) is colored {5,7} (with a sum of 12) from the first and second segments. - [4,7) is colored {1,11} (with a sum of 12) from the third and fourth segments. Note that returning a single segment [1,7) is incorrect because the mixed color sets are different.

Constraints:

1 <= segments.length <= 2 * 10

4

segments[i].length == 3

1 <= start

i

< end

i

<= 10

5

1 <= color

i

<= 10

9

Each

color

i

is distinct.

Code Snippets

C++:

```
class Solution {
public:
vector<vector<long long>> splitPainting(vector<vector<int>>& segments) {

}
};
```

Java:

```
class Solution {
public List<List<Long>> splitPainting(int[][] segments) {

}
}
```

Python3:

```
class Solution:
def splitPainting(self, segments: List[List[int]]) -> List[List[int]]:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function splitPainting(segments: number[][]): number[][] {

};
```

C#:

```
public class Solution {
public IList<IList<long>> SplitPainting(int[][] segments) {

}
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
long long** splitPainting(int** segments, int segmentsSize, int*
segmentsColSize, int* returnSize, int** returnColumnSizes) {

}
```

Go:

```
func splitPainting(segments [][]int) [][]int64 {

}
```

Kotlin:

```
class Solution {
fun splitPainting(segments: Array<IntArray>): List<List<Long>> {

}
}
```

Swift:

```
class Solution {
func splitPainting(_ segments: [[Int]]) -> [[Int]] {
```

```
}
}
```

Rust:

```
impl Solution {
pub fn split_painting(segments: Vec<Vec<i32>>) -> Vec<Vec<i64>> {

}
}
```

Ruby:

```
# @param {Integer[][]} segments
# @return {Integer[][]}
def split_painting(segments)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $segments
* @return Integer[][]
*/
function splitPainting($segments) {

}
}
```

Dart:

```
class Solution {
List<List<int>> splitPainting(List<List<int>> segments) {

}
}
```

Scala:

```
object Solution {
def splitPainting(segments: Array[Array[Int]]): List[List[Long]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec split_painting(segments :: [[integer]]) :: [[integer]]
def split_painting(segments) do

end
end
```

Erlang:

```
-spec split_painting(Segments :: [[integer()]]) -> [[integer()]].
split_painting(Segments) ->
.
```

Racket:

```
(define/contract (split-painting segments)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Describe the Painting
* Difficulty: Medium
* Tags: array, hash, sort
*
* 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:
vector<vector<long long>> splitPainting(vector<vector<int>>& segments) {

}
};
```

Java Solution:

```
/**
* Problem: Describe the Painting
* Difficulty: Medium
* Tags: array, hash, sort
*
* 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 List<List<Long>> splitPainting(int[][] segments) {

}
}
```

Python3 Solution:

```
"""
Problem: Describe the Painting
Difficulty: Medium
Tags: array, hash, sort

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 splitPainting(self, segments: List[List[int]]) -> List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Describe the Painting
* Difficulty: Medium
* Tags: array, hash, sort
*
* 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[][]} segments
* @return {number[][]}
*/
var splitPainting = function(segments) {

};
```

TypeScript Solution:

```
/**
* Problem: Describe the Painting
* Difficulty: Medium
* Tags: array, hash, sort
*
* 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 splitPainting(segments: number[][]): number[][] {
```

```
};
```

C# Solution:

```
/*
* Problem: Describe the Painting
* Difficulty: Medium
* Tags: array, hash, sort
*
* 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 IList<IList<long>> SplitPainting(int[][] segments) {

}
}
```

C Solution:

```
/*
* Problem: Describe the Painting
* Difficulty: Medium
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
long long** splitPainting(int** segments, int segmentsSize, int*
segmentsColSize, int* returnSize, int** returnColumnSizes) {
```

```
}
```

Go Solution:

```
// Problem: Describe the Painting
// Difficulty: Medium
// Tags: array, hash, sort
//
// 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 splitPainting(segments [][]int) [][]int64 {

}
```

Kotlin Solution:

```
class Solution {
fun splitPainting(segments: Array<IntArray>): List<List<Long>> {

}
}
```

Swift Solution:

```
class Solution {
func splitPainting(_ segments: [[Int]]) -> [[Int]] {

}
}
```

Rust Solution:

```
// Problem: Describe the Painting
// Difficulty: Medium
// Tags: array, hash, sort
//
// 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 split_painting(segments: Vec<Vec<i32>>) -> Vec<Vec<i64>> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} segments
# @return {Integer[][]}
def split_painting(segments)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $segments
* @return Integer[][]
*/
function splitPainting($segments) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> splitPainting(List<List<int>> segments) {

}
}
```

Scala Solution:

```
object Solution {
def splitPainting(segments: Array[Array[Int]]): List[List[Long]] = {
```

```
}
}
```

Elixir Solution:

```
defmodule Solution do
@spec split_painting(segments :: [[integer]]) :: [[integer]]
def split_painting(segments) do

end
end
```

Erlang Solution:

```
-spec split_painting(Segments :: [[integer()]]) -> [[integer()]].
split_painting(Segments) ->
.
```

Racket Solution:

```
(define/contract (split-painting segments)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)
```