Problem 3414: Maximum Score of Non-overlapping Intervals

Problem Information

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

Problem Description

You are given a 2D integer array

intervals

, where

intervals[i] = [l

i

, r

i

, weight

i

]

. Interval

i

starts at position

l

i

and ends at

r

i

, and has a weight of

weight

i

. You can choose

up to

4

non-overlapping

intervals. The

score

of the chosen intervals is defined as the total sum of their weights.

Return the

lexicographically smallest

array of at most 4 indices from

intervals

with

maximum

score, representing your choice of non-overlapping intervals.

Two intervals are said to be

non-overlapping

if they do not share any points. In particular, intervals sharing a left or right boundary are considered overlapping.

Example 1:

Input:

intervals = [[1,3,2],[4,5,2],[1,5,5],[6,9,3],[6,7,1],[8,9,1]]

Output:

[2,3]

Explanation:

You can choose the intervals with indices 2, and 3 with respective weights of 5, and 3.

Example 2:

Input:

intervals = [[5,8,1],[6,7,7],[4,7,3],[9,10,6],[7,8,2],[11,14,3],[3,5,5]]

Output:

[1,3,5,6]

Explanation:

You can choose the intervals with indices 1, 3, 5, and 6 with respective weights of 7, 6, 3, and 5.

Constraints:

1 <= intevals.length <= 5 * 10

4

intervals[i].length == 3

intervals[i] = [l

i

, r

i

, weight

i

]

1 <= l

i

<= r

i

<= 10

9

1 <= weight

i

<= 10

9

Code Snippets

C++:

```
class Solution {
public:
vector<int> maximumWeight(vector<vector<int>>& intervals) {

}
};
```

Java:

```
class Solution {
public int[] maximumWeight(List<List<Integer>> intervals) {

}
}
```

Python3:

```
class Solution:
def maximumWeight(self, intervals: List[List[int]]) -> List[int]:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maximumWeight(intervals: number[][]): number[] {

};
```

C#:

```
public class Solution {
public int[] MaximumWeight(IList<IList<int>> intervals) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* maximumWeight(int** intervals, int intervalsSize, int* intervalsColSize,
int* returnSize) {

}
```

Go:

```
func maximumWeight(intervals [][]int) []int {

}
```

Kotlin:

```
class Solution {
fun maximumWeight(intervals: List<List<Int>>): IntArray {
}
}
```

Swift:

```
class Solution {
func maximumWeight(_ intervals: [[Int]]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn maximum_weight(intervals: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[][]} intervals
# @return {Integer[]}
def maximum_weight(intervals)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $intervals
* @return Integer[]
*/
function maximumWeight($intervals) {

}
}
```

Dart:

```
class Solution {
List<int> maximumWeight(List<List<int>> intervals) {

}
}
```

Scala:

```
object Solution {
def maximumWeight(intervals: List[List[Int]]): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec maximum_weight(intervals :: [[integer]]) :: [integer]
def maximum_weight(intervals) do

end
end
```

Erlang:

```
-spec maximum_weight(Intervals :: [[integer()]]) -> [integer()].
maximum_weight(Intervals) ->
.
```

Racket:

```
(define/contract (maximum-weight intervals)
(-> (listof (listof exact-integer?)) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Score of Non-overlapping Intervals
* Difficulty: Hard
* Tags: array, graph, dp, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
vector<int> maximumWeight(vector<vector<int>>& intervals) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Score of Non-overlapping Intervals
* Difficulty: Hard
* Tags: array, graph, dp, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int[] maximumWeight(List<List<Integer>> intervals) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Score of Non-overlapping Intervals
Difficulty: Hard
Tags: array, graph, dp, sort, search

Approach: Use two pointers or sliding window technique
```

```
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Score of Non-overlapping Intervals
* Difficulty: Hard
* Tags: array, graph, dp, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[][]} intervals
* @return {number[]}
*/
var maximumWeight = function(intervals) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Score of Non-overlapping Intervals
* Difficulty: Hard
* Tags: array, graph, dp, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function maximumWeight(intervals: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Maximum Score of Non-overlapping Intervals
* Difficulty: Hard
* Tags: array, graph, dp, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int[] MaximumWeight(IList<IList<int>> intervals) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Score of Non-overlapping Intervals
* Difficulty: Hard
* Tags: array, graph, dp, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
```

```
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* maximumWeight(int** intervals, int intervalsSize, int* intervalsColSize,
int* returnSize) {

}
```

Go Solution:

```
// Problem: Maximum Score of Non-overlapping Intervals
// Difficulty: Hard
// Tags: array, graph, dp, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func maximumWeight(intervals [][]int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun maximumWeight(intervals: List<List<Int>>): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func maximumWeight(_ intervals: [[Int]]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Maximum Score of Non-overlapping Intervals
// Difficulty: Hard
// Tags: array, graph, dp, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn maximum_weight(intervals: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} intervals
# @return {Integer[]}
def maximum_weight(intervals)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $intervals
* @return Integer[]
*/
function maximumWeight($intervals) {

}
}
```

Dart Solution:

```
class Solution {
List<int> maximumWeight(List<List<int>> intervals) {

}
}
```

Scala Solution:

```
object Solution {
def maximumWeight(intervals: List[List[Int]]): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximum_weight(intervals :: [[integer]]) :: [integer]
def maximum_weight(intervals) do

end
end
```

Erlang Solution:

```
-spec maximum_weight(Intervals :: [[integer()]]) -> [integer()].
maximum_weight(Intervals) ->
.
```

Racket Solution:

```
(define/contract (maximum-weight intervals)
(-> (listof (listof exact-integer?)) (listof exact-integer?))
)
```