Problem 3413: Maximum Coins From K Consecutive Bags

Problem Information

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

Problem Description

There are an infinite amount of bags on a number line, one bag for each coordinate. Some of these bags contain coins.

You are given a 2D array

coins

, where

coins[i] = [l

i

, r

i

, c

i

]

denotes that every bag from

l

i

to

r

i

contains

c

i

coins.

The segments that

coins

contain are non-overlapping.

You are also given an integer

k

.

Return the

maximum

amount of coins you can obtain by collecting

k

consecutive bags.

Example 1:

Input:

coins = [[8,10,1],[1,3,2],[5,6,4]], k = 4

Output:

10

Explanation:

Selecting bags at positions

[3, 4, 5, 6]

gives the maximum number of coins:

2 + 0 + 4 + 4 = 10

.

Example 2:

Input:

coins = [[1,10,3]], k = 2

Output:

6

Explanation:

Selecting bags at positions

[1, 2]

gives the maximum number of coins:

3 + 3 = 6

.

Constraints:

1 <= coins.length <= 10

5

1 <= k <= 10

9

coins[i] == [l

i

, r

i

, c

i

]

1 <= l

i

<= r

i

<= 10

9

1 <= c

i

<= 1000

The given segments are non-overlapping.

Code Snippets

C++:

```
class Solution {
public:
long long maximumCoins(vector<vector<int>>& coins, int k) {

}
};
```

Java:

```
class Solution {
public long maximumCoins(int[][] coins, int k) {

}
}
```

Python3:

```
class Solution:
def maximumCoins(self, coins: List[List[int]], k: int) -> int:
```

Python:

```
class Solution(object):
def maximumCoins(self, coins, k):
"""
:type coins: List[List[int]]
```

```
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} coins
* @param {number} k
* @return {number}
*/
var maximumCoins = function(coins, k) {

};
```

TypeScript:

```
function maximumCoins(coins: number[][], k: number): number {

};
```

C#:

```
public class Solution {
public long MaximumCoins(int[][] coins, int k) {

}
}
```

C:

```
long long maximumCoins(int** coins, int coinsSize, int* coinsColSize, int k)
{

}
```

Go:

```
func maximumCoins(coins [][]int, k int) int64 {

}
```

Kotlin:

```
class Solution {
fun maximumCoins(coins: Array<IntArray>, k: Int): Long {

}
}
```

Swift:

```
class Solution {
func maximumCoins(_ coins: [[Int]], _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn maximum_coins(coins: Vec<Vec<i32>>, k: i32) -> i64 {

}
}
```

Ruby:

```
# @param {Integer[][]} coins
# @param {Integer} k
# @return {Integer}
def maximum_coins(coins, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $coins
* @param Integer $k
* @return Integer
*/
function maximumCoins($coins, $k) {
```

```
}
}
```

Dart:

```
class Solution {
int maximumCoins(List<List<int>> coins, int k) {

}
}
```

Scala:

```
object Solution {
def maximumCoins(coins: Array[Array[Int]], k: Int): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec maximum_coins(coins :: [[integer]], k :: integer) :: integer
def maximum_coins(coins, k) do

end
end
```

Erlang:

```
-spec maximum_coins(Coins :: [[integer()]], K :: integer()) -> integer().
maximum_coins(Coins, K) ->
.
```

Racket:

```
(define/contract (maximum-coins coins k)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Coins From K Consecutive Bags
* 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:
long long maximumCoins(vector<vector<int>>& coins, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Coins From K Consecutive Bags
* 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 long maximumCoins(int[][] coins, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Coins From K Consecutive Bags
```

```
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 maximumCoins(self, coins: List[List[int]], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maximumCoins(self, coins, k):
"""
:type coins: List[List[int]]
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Coins From K Consecutive Bags
* 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[][]} coins
* @param {number} k
* @return {number}
*/
var maximumCoins = function(coins, k) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Maximum Coins From K Consecutive Bags
* 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 maximumCoins(coins: number[][], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Coins From K Consecutive Bags
* 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 long MaximumCoins(int[][] coins, int k) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Coins From K Consecutive Bags
```

```
* 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
*/

long long maximumCoins(int** coins, int coinsSize, int* coinsColSize, int k)
{

}
```

Go Solution:

```
// Problem: Maximum Coins From K Consecutive Bags
// 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 maximumCoins(coins [][]int, k int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun maximumCoins(coins: Array<IntArray>, k: Int): Long {

}
}
```

Swift Solution:

```
class Solution {
func maximumCoins(_ coins: [[Int]], _ k: Int) -> Int {

}
```

```
}
```

Rust Solution:

```
// Problem: Maximum Coins From K Consecutive Bags
// 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_coins(coins: Vec<Vec<i32>>, k: i32) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} coins
# @param {Integer} k
# @return {Integer}
def maximum_coins(coins, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $coins
* @param Integer $k
* @return Integer
*/
function maximumCoins($coins, $k) {

}
}
```

Dart Solution:

```
class Solution {
int maximumCoins(List<List<int>> coins, int k) {

}
}
```

Scala Solution:

```
object Solution {
def maximumCoins(coins: Array[Array[Int]], k: Int): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximum_coins(coins :: [[integer]], k :: integer) :: integer
def maximum_coins(coins, k) do

end
end
```

Erlang Solution:

```
-spec maximum_coins(Coins :: [[integer()]], K :: integer()) -> integer().
maximum_coins(Coins, K) ->
.
```

Racket Solution:

```
(define/contract (maximum-coins coins k)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```