

Problem 1751: Maximum Number of Events That Can Be Attended II

Problem Information

Difficulty: Hard

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given an array of

events

where

events[i] = [startDay

i

, endDay

i

, value

i

]

. The

i

th

event starts at

startDay

i

and ends at

endDay

i

, and if you attend this event, you will receive a value of

value

i

. You are also given an integer

k

which represents the maximum number of events you can attend.

You can only attend one event at a time. If you choose to attend an event, you must attend the

entire

event. Note that the end day is

inclusive

: that is, you cannot attend two events where one of them starts and the other ends on the same day.

Return

the

maximum sum

of values that you can receive by attending events.

Example 1:

Time	1	2	3	4
Event 0	4			
Event 1			3	
Event 2		1		

Input:

events = [[1,2,4],[3,4,3],[2,3,1]], k = 2

Output:

7

Explanation:

Choose the green events, 0 and 1 (0-indexed) for a total value of $4 + 3 = 7$.

Example 2:

Time	1	2	3	4
Event 0	4			
Event 1			3	
Event 2		10		

Input:

events = [[1,2,4],[3,4,3],[2,3,10]], k = 2

Output:

10

Explanation:

Choose event 2 for a total value of 10. Notice that you cannot attend any other event as they overlap, and that you do

not

have to attend k events.

Example 3:

Time	1	2	3	4
Event 0	1			
Event 1		2		
Event 2			3	
Event 3				4

Input:

events = [[1,1,1],[2,2,2],[3,3,3],[4,4,4]], k = 3

Output:

9

Explanation:

Although the events do not overlap, you can only attend 3 events. Pick the highest valued three.

Constraints:

$1 \leq k \leq \text{events.length}$

$1 \leq k * \text{events.length} \leq 10$

6

1 <= startDay

i

<= endDay

i

<= 10

9

1 <= value

i

<= 10

6

Code Snippets

C++:

```
class Solution {
public:
    int maxValue(vector<vector<int>>& events, int k) {
        }
    };
}
```

Java:

```
class Solution {
public int maxValue(int[][][] events, int k) {
    }
}
```

```
}
```

Python3:

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

Python:

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

JavaScript:

```
/**  
 * @param {number[][]} events  
 * @param {number} k  
 * @return {number}  
 */  
var maxValue = function(events, k) {  
  
};
```

TypeScript:

```
function maxValue(events: number[][], k: number): number {  
  
};
```

C#:

```
public class Solution {  
    public int MaxValue(int[][] events, int k) {  
  
    }  
}
```

C:

```
int maxValue(int** events, int eventsSize, int* eventsColSize, int k) {  
  
}
```

Go:

```
func maxValue(events [][]int, k int) int {  
  
}
```

Kotlin:

```
class Solution {  
    fun maxValue(events: Array<IntArray>, k: Int): Int {  
  
    }  
}
```

Swift:

```
class Solution {  
    func maxValue(_ events: [[Int]], _ k: Int) -> Int {  
  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn max_value(events: Vec<Vec<i32>>, k: i32) -> i32 {  
  
    }  
}
```

Ruby:

```
# @param {Integer[][]} events  
# @param {Integer} k  
# @return {Integer}  
def max_value(events, k)
```

```
end
```

PHP:

```
class Solution {

    /**
     * @param Integer[][] $events
     * @param Integer $k
     * @return Integer
     */
    function maxValue($events, $k) {

    }
}
```

Dart:

```
class Solution {
  int maxValue(List<List<int>> events, int k) {
    }
}
```

Scala:

```
object Solution {
  def maxValue(events: Array[Array[Int]], k: Int): Int = {
    }
}
```

Elixir:

```
defmodule Solution do
  @spec max_value(events :: [[integer]], k :: integer) :: integer
  def max_value(events, k) do
    end
  end
end
```

Erlang:

```
-spec max_value(Events :: [[integer()]], K :: integer()) -> integer().  
max_value(Events, K) ->  
.
```

Racket:

```
(define/contract (max-value events k)  
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)  
)
```

Solutions

C++ Solution:

```
/*  
 * Problem: Maximum Number of Events That Can Be Attended II  
 * Difficulty: Hard  
 * Tags: array, 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 maxValue(vector<vector<int>>& events, int k) {  
  
    }  
};
```

Java Solution:

```
/**  
 * Problem: Maximum Number of Events That Can Be Attended II  
 * Difficulty: Hard  
 * Tags: array, 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 maxValue(int[][] events, int k) {
}

}
}

```

Python3 Solution:

```

"""
Problem: Maximum Number of Events That Can Be Attended II
Difficulty: Hard
Tags: array, 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 maxValue(self, events: List[List[int]], k: int) -> int:
    # TODO: Implement optimized solution
    pass

```

Python Solution:

```

class Solution(object):
    def maxValue(self, events, k):
        """
        :type events: List[List[int]]
        :type k: int
        :rtype: int
        """

```

JavaScript Solution:

```

/**
 * Problem: Maximum Number of Events That Can Be Attended II
 * Difficulty: Hard
 * Tags: array, 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[][]} events
 * @param {number} k
 * @return {number}
 */
var maxValue = function(events, k) {

};

```

TypeScript Solution:

```

/**
 * Problem: Maximum Number of Events That Can Be Attended II
 * Difficulty: Hard
 * Tags: array, 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 maxValue(events: number[][], k: number): number {

};

```

C# Solution:

```

/*
 * Problem: Maximum Number of Events That Can Be Attended II
 * Difficulty: Hard
 * Tags: array, 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 MaxValue(int[][] events, int k) {

}
}

```

C Solution:

```

/*
 * Problem: Maximum Number of Events That Can Be Attended II
 * Difficulty: Hard
 * Tags: array, 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
 */

int maxValue(int** events, int eventsSize, int* eventsColSize, int k) {

}

```

Go Solution:

```

// Problem: Maximum Number of Events That Can Be Attended II
// Difficulty: Hard
// Tags: array, 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 maxValue(events [][]int, k int) int {

}

```

Kotlin Solution:

```
class Solution {  
    fun maxValue(events: Array<IntArray>, k: Int): Int {  
        }  
        }  
}
```

Swift Solution:

```
class Solution {  
    func maxValue(_ events: [[Int]], _ k: Int) -> Int {  
        }  
        }  
}
```

Rust Solution:

```
// Problem: Maximum Number of Events That Can Be Attended II  
// Difficulty: Hard  
// Tags: array, 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 max_value(events: Vec<Vec<i32>>, k: i32) -> i32 {  
        }  
        }  
}
```

Ruby Solution:

```
# @param {Integer[][]} events  
# @param {Integer} k  
# @return {Integer}  
def max_value(events, k)  
  
end
```

PHP Solution:

```
class Solution {

    /**
     * @param Integer[][] $events
     * @param Integer $k
     * @return Integer
     */
    function maxValue($events, $k) {

    }
}
```

Dart Solution:

```
class Solution {
int maxValue(List<List<int>> events, int k) {

}
```

Scala Solution:

```
object Solution {
def maxValue(events: Array[Array[Int]], k: Int): Int = {

}
```

Elixir Solution:

```
defmodule Solution do
@spec max_value(events :: [[integer]], k :: integer) :: integer
def max_value(events, k) do

end
end
```

Erlang Solution:

```
-spec max_value(Events :: [[integer()]], K :: integer()) -> integer().
max_value(Events, K) ->
.
```

Racket Solution:

```
(define/contract (max-value events k)
  (-> (listof (listof exact-integer?)) exact-integer? exact-integer?))
)
```