Problem 1024: Video Stitching

Problem Information

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

Problem Description

You are given a series of video clips from a sporting event that lasted

time

seconds. These video clips can be overlapping with each other and have varying lengths.

Each video clip is described by an array

clips

where

clips[i] = [start

i

, end

i

]

indicates that the ith clip started at

start

i

and ended at

end

i

.

We can cut these clips into segments freely.

For example, a clip

[0, 7]

can be cut into segments

[0, 1] + [1, 3] + [3, 7]

.

Return

the minimum number of clips needed so that we can cut the clips into segments that cover the entire sporting event

[0, time]

. If the task is impossible, return

-1

.

Example 1:

Input:

clips = [[0,2],[4,6],[8,10],[1,9],[1,5],[5,9]], time = 10

Output:

3

Explanation:

We take the clips [0,2], [8,10], [1,9]; a total of 3 clips. Then, we can reconstruct the sporting event as follows: We cut [1,9] into segments [1,2] + [2,8] + [8,9]. Now we have segments [0,2] + [2,8] + [8,10] which cover the sporting event [0, 10].

Example 2:

Input:

clips = [[0,1],[1,2]], time = 5

Output:

-1

Explanation:

We cannot cover [0,5] with only [0,1] and [1,2].

Example 3:

Input:

clips = [[0,1],[6,8],[0,2],[5,6],[0,4],[0,3],[6,7],[1,3],[4,7],[1,4],[2,5],[2,6],[3,4],[4,5],[5,7],[6,9]], time = 9

Output:

3

Explanation:

We can take clips [0,4], [4,7], and [6,9].

Constraints:

1 <= clips.length <= 100

0 <= start

i

<= end

i

<= 100

1 <= time <= 100

Code Snippets

C++:

```
class Solution {
public:
int videoStitching(vector<vector<int>>& clips, int time) {

}
};
```

Java:

```
class Solution {
public int videoStitching(int[][] clips, int time) {

}
}
```

Python3:

```
class Solution:
def videoStitching(self, clips: List[List[int]], time: int) -> int:
```

Python:

```
class Solution(object):
def videoStitching(self, clips, time):
"""
:type clips: List[List[int]]
:type time: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} clips
* @param {number} time
* @return {number}
*/
var videoStitching = function(clips, time) {

};
```

TypeScript:

```
function videoStitching(clips: number[][], time: number): number {

};
```

C#:

```
public class Solution {
public int VideoStitching(int[][] clips, int time) {

}
}
```

C:

```
int videoStitching(int** clips, int clipsSize, int* clipsColSize, int time) {

}
```

Go:

```
func videoStitching(clips [][]int, time int) int {

}
```

Kotlin:

```
class Solution {
fun videoStitching(clips: Array<IntArray>, time: Int): Int {

}
}
```

Swift:

```
class Solution {
func videoStitching(_ clips: [[Int]], _ time: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn video_stitching(clips: Vec<Vec<i32>>, time: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} clips
# @param {Integer} time
# @return {Integer}
def video_stitching(clips, time)

end
```

PHP:

```
class Solution {
```

```
/**
* @param Integer[][] $clips
* @param Integer $time
* @return Integer
*/
function videoStitching($clips, $time) {

}
}
```

Dart:

```
class Solution {
int videoStitching(List<List<int>> clips, int time) {

}
}
```

Scala:

```
object Solution {
def videoStitching(clips: Array[Array[Int]], time: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec video_stitching(clips :: [[integer]], time :: integer) :: integer
def video_stitching(clips, time) do

end
end
```

Erlang:

```
-spec video_stitching(Clips :: [[integer()]], Time :: integer()) ->
integer().
video_stitching(Clips, Time) ->
.
```

Racket:

```
(define/contract (video-stitching clips time)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Video Stitching
* Difficulty: Medium
* Tags: array, dp, greedy
*
* 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 videoStitching(vector<vector<int>>& clips, int time) {

}
};
```

Java Solution:

```
/**
* Problem: Video Stitching
* Difficulty: Medium
* Tags: array, dp, greedy
*
* 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 videoStitching(int[][] clips, int time) {
}
}
```

Python3 Solution:

```
"""
Problem: Video Stitching
Difficulty: Medium
Tags: array, dp, greedy

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 videoStitching(self, clips: List[List[int]], time: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def videoStitching(self, clips, time):
"""
:type clips: List[List[int]]
:type time: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Video Stitching
* Difficulty: Medium
* Tags: array, dp, greedy
*
* 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[][]} clips
* @param {number} time
* @return {number}
*/
var videoStitching = function(clips, time) {

};
```

TypeScript Solution:

```
/**
* Problem: Video Stitching
* Difficulty: Medium
* Tags: array, dp, greedy
*
* 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 videoStitching(clips: number[][], time: number): number {

};
```

C# Solution:

```
/*
* Problem: Video Stitching
* Difficulty: Medium
* Tags: array, dp, greedy
*
* 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 VideoStitching(int[][] clips, int time) {
}
}
```

C Solution:

```
/*
* Problem: Video Stitching
* Difficulty: Medium
* Tags: array, dp, greedy
*
* 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 videoStitching(int** clips, int clipsSize, int* clipsColSize, int time) {

}
```

Go Solution:

```
// Problem: Video Stitching
// Difficulty: Medium
// Tags: array, dp, greedy
//
// 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 videoStitching(clips [][]int, time int) int {

}
```

Kotlin Solution:

```
class Solution {
fun videoStitching(clips: Array<IntArray>, time: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func videoStitching(_ clips: [[Int]], _ time: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Video Stitching
// Difficulty: Medium
// Tags: array, dp, greedy
//
// 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 video_stitching(clips: Vec<Vec<i32>>, time: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} clips
# @param {Integer} time
# @return {Integer}
def video_stitching(clips, time)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $clips
* @param Integer $time
* @return Integer
*/
function videoStitching($clips, $time) {

}
}
```

Dart Solution:

```
class Solution {
int videoStitching(List<List<int>> clips, int time) {

}
}
```

Scala Solution:

```
object Solution {
def videoStitching(clips: Array[Array[Int]], time: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec video_stitching(clips :: [[integer]], time :: integer) :: integer
def video_stitching(clips, time) do

end
end
```

Erlang Solution:

```
-spec video_stitching(Clips :: [[integer()]], Time :: integer()) ->
integer().
video_stitching(Clips, Time) ->
.
```

Racket Solution:

```
(define/contract (video-stitching clips time)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)

)
```