Problem 3480: Maximize Subarrays After Removing One Conflicting Pair

Problem Information

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

Problem Description

You are given an integer

n

which represents an array

nums

containing the numbers from 1 to

n

in order. Additionally, you are given a 2D array

conflictingPairs

, where

conflictingPairs[i] = [a, b]

indicates that

a

and

b

form a conflicting pair.

Remove

exactly

one element from

conflictingPairs

. Afterward, count the number of

non-empty subarrays

of

nums

which do not contain both

a

and

b

for any remaining conflicting pair

[a, b]

.

Return the

maximum

number of subarrays possible after removing

exactly

one conflicting pair.

Example 1:

Input:

n = 4, conflictingPairs = [[2,3],[1,4]]

Output:

9

Explanation:

Remove

[2, 3]

from

conflictingPairs

. Now,

conflictingPairs = [[1, 4]]

.

There are 9 subarrays in

nums

where

[1, 4]

do not appear together. They are

[1]

,

[2]

,

[3]

,

[4]

,

[1, 2]

,

[2, 3]

,

[3, 4]

,

[1, 2, 3]

and

[2, 3, 4]

.

The maximum number of subarrays we can achieve after removing one element from

conflictingPairs

is 9.

Example 2:

Input:

n = 5, conflictingPairs = [[1,2],[2,5],[3,5]]

Output:

12

Explanation:

Remove

[1, 2]

from

conflictingPairs

. Now,

conflictingPairs = [[2, 5], [3, 5]]

.

There are 12 subarrays in

nums

where

[2, 5]

and

[3, 5]

do not appear together.

The maximum number of subarrays we can achieve after removing one element from

conflictingPairs

is 12.

Constraints:

2 <= n <= 10

5

1 <= conflictingPairs.length <= 2 * n

conflictingPairs[i].length == 2

1 <= conflictingPairs[i][j] <= n

conflictingPairs[i][0] != conflictingPairs[i][1]

Code Snippets

C++:

```
class Solution {
public:
long long maxSubarrays(int n, vector<vector<int>>& conflictingPairs) {

}
};
```

Java:

```
class Solution {
public long maxSubarrays(int n, int[][] conflictingPairs) {

}
}
```

Python3:

```
class Solution:
def maxSubarrays(self, n: int, conflictingPairs: List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def maxSubarrays(self, n, conflictingPairs):
"""
:type n: int
:type conflictingPairs: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} conflictingPairs
* @return {number}
*/
var maxSubarrays = function(n, conflictingPairs) {

};
```

TypeScript:

```
function maxSubarrays(n: number, conflictingPairs: number[][]): number {

};
```

C#:

```
public class Solution {
public long MaxSubarrays(int n, int[][] conflictingPairs) {
```

```
}
}
```

C:

```
long long maxSubarrays(int n, int** conflictingPairs, int
conflictingPairsSize, int* conflictingPairsColSize) {

}
```

Go:

```
func maxSubarrays(n int, conflictingPairs [][]int) int64 {

}
```

Kotlin:

```
class Solution {
fun maxSubarrays(n: Int, conflictingPairs: Array<IntArray>): Long {

}
}
```

Swift:

```
class Solution {
func maxSubarrays(_ n: Int, _ conflictingPairs: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_subarrays(n: i32, conflicting_pairs: Vec<Vec<i32>>) -> i64 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} conflicting_pairs
# @return {Integer}
def max_subarrays(n, conflicting_pairs)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $conflictingPairs
* @return Integer
*/
function maxSubarrays($n, $conflictingPairs) {

}
}
```

Dart:

```
class Solution {
int maxSubarrays(int n, List<List<int>> conflictingPairs) {

}
}
```

Scala:

```
object Solution {
def maxSubarrays(n: Int, conflictingPairs: Array[Array[Int]]): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_subarrays(n :: integer, conflicting_pairs :: [[integer]]) ::
integer
def max_subarrays(n, conflicting_pairs) do
```

```
end
end
```

Erlang:

```
-spec max_subarrays(N :: integer(), ConflictingPairs :: [[integer()]]) ->
integer().
max_subarrays(N, ConflictingPairs) ->
.
```

Racket:

```
(define/contract (max-subarrays n conflictingPairs)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximize Subarrays After Removing One Conflicting Pair
* Difficulty: Hard
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
long long maxSubarrays(int n, vector<vector<int>>& conflictingPairs) {

}
};
```

Java Solution:

```
/**
* Problem: Maximize Subarrays After Removing One Conflicting Pair
* Difficulty: Hard
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public long maxSubarrays(int n, int[][] conflictingPairs) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximize Subarrays After Removing One Conflicting Pair
Difficulty: Hard
Tags: array, tree

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

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

Python Solution:

```
class Solution(object):
def maxSubarrays(self, n, conflictingPairs):
"""
:type n: int
:type conflictingPairs: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximize Subarrays After Removing One Conflicting Pair
* Difficulty: Hard
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {number} n
* @param {number[][]} conflictingPairs
* @return {number}
*/
var maxSubarrays = function(n, conflictingPairs) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximize Subarrays After Removing One Conflicting Pair
* Difficulty: Hard
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function maxSubarrays(n: number, conflictingPairs: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Maximize Subarrays After Removing One Conflicting Pair
* Difficulty: Hard
```

```
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public long MaxSubarrays(int n, int[][] conflictingPairs) {

}
}
```

C Solution:

```
/*
* Problem: Maximize Subarrays After Removing One Conflicting Pair
* Difficulty: Hard
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

long long maxSubarrays(int n, int** conflictingPairs, int
conflictingPairsSize, int* conflictingPairsColSize) {

}
```

Go Solution:

```
// Problem: Maximize Subarrays After Removing One Conflicting Pair
// Difficulty: Hard
// Tags: array, tree
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func maxSubarrays(n int, conflictingPairs [][]int) int64 {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun maxSubarrays(n: Int, conflictingPairs: Array<IntArray>): Long {

}
}
```

Swift Solution:

```
class Solution {
func maxSubarrays(_ n: Int, _ conflictingPairs: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximize Subarrays After Removing One Conflicting Pair
// Difficulty: Hard
// Tags: array, tree
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn max_subarrays(n: i32, conflicting_pairs: Vec<Vec<i32>>) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} conflicting_pairs
# @return {Integer}
def max_subarrays(n, conflicting_pairs)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $conflictingPairs
* @return Integer
*/
function maxSubarrays($n, $conflictingPairs) {

}
}
```

Dart Solution:

```
class Solution {
int maxSubarrays(int n, List<List<int>> conflictingPairs) {

}
}
```

Scala Solution:

```
object Solution {
def maxSubarrays(n: Int, conflictingPairs: Array[Array[Int]]): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_subarrays(n :: integer, conflicting_pairs :: [[integer]]) ::
integer
def max_subarrays(n, conflicting_pairs) do

end
```

```
end
```

Erlang Solution:

```
-spec max_subarrays(N :: integer(), ConflictingPairs :: [[integer()]]) ->
integer().
max_subarrays(N, ConflictingPairs) ->
.
```

Racket Solution:

```
(define/contract (max-subarrays n conflictingPairs)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```