

Problem 2251: Number of Flowers in Full Bloom

Problem Information

Difficulty: Hard

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given a

0-indexed

2D integer array

flowers

, where

`flowers[i] = [start`

`i`

`, end`

`i`

`]`

means the

`i`

th

flower will be in

full bloom

from

start

i

to

end

i

(

inclusive

). You are also given a

0-indexed

integer array

people

of size

n

, where

people[i]

is the time that the

i

th

person will arrive to see the flowers.

Return

an integer array

answer

of size

n

, where

answer[i]

is the

number

of flowers that are in full bloom when the

i

th

person arrives.

Example 1:

1	2	3	4	5	6	7	8	9	10	11	12	13
	0	1				2				3		

Input:

flowers = [[1,6],[3,7],[9,12],[4,13]], people = [2,3,7,11]

Output:

[1,2,2,2]

Explanation:

The figure above shows the times when the flowers are in full bloom and when the people arrive. For each person, we return the number of flowers in full bloom during their arrival.

Example 2:

1	2	3	4	5	6	7	8	9	10
	2	0 1							

Input:

flowers = [[1,10],[3,3]], people = [3,3,2]

Output:

[2,2,1]

Explanation:

The figure above shows the times when the flowers are in full bloom and when the people arrive. For each person, we return the number of flowers in full bloom during their arrival.

Constraints:

$1 \leq \text{flowers.length} \leq 5 * 10$

4

$\text{flowers}[i].length == 2$

$1 \leq \text{start}$

i

$\leq \text{end}$

i

≤ 10

9

$1 \leq \text{people.length} \leq 5 * 10$

4

$1 \leq \text{people}[i] \leq 10$

9

Code Snippets

C++:

```
class Solution {  
public:  
vector<int> fullBloomFlowers(vector<vector<int>>& flowers, vector<int>&  
people) {  
  
}  
};
```

Java:

```
class Solution {  
public int[] fullBloomFlowers(int[][] flowers, int[] people) {  
  
}  
}
```

Python3:

```
class Solution:  
def fullBloomFlowers(self, flowers: List[List[int]], people: List[int]) ->  
List[int]:
```

Python:

```
class Solution(object):  
def fullBloomFlowers(self, flowers, people):  
    """  
    :type flowers: List[List[int]]  
    :type people: List[int]  
    :rtype: List[int]  
    """
```

JavaScript:

```
/**  
 * @param {number[][]} flowers  
 * @param {number[]} people  
 * @return {number[]}  
 */  
var fullBloomFlowers = function(flowers, people) {
```

```
};
```

TypeScript:

```
function fullBloomFlowers(flowers: number[][][], people: number[]): number[] {  
}  
};
```

C#:

```
public class Solution {  
    public int[] FullBloomFlowers(int[][][] flowers, int[] people) {  
          
    }  
}
```

C:

```
/**  
 * Note: The returned array must be malloced, assume caller calls free().  
 */  
int* fullBloomFlowers(int** flowers, int flowersSize, int* flowersColSize,  
int* people, int peopleSize, int* returnSize) {  
  
}
```

Go:

```
func fullBloomFlowers(flowers [][]int, people []int) []int {  
}  
}
```

Kotlin:

```
class Solution {  
    fun fullBloomFlowers(flowers: Array<IntArray>, people: IntArray): IntArray {  
          
    }  
}
```

Swift:

```
class Solution {  
func fullBloomFlowers(_ flowers: [[Int]], _ people: [Int]) -> [Int] {  
  
}  
}  
}
```

Rust:

```
impl Solution {  
pub fn full_bloom_flowers(flowers: Vec<Vec<i32>>, people: Vec<i32>) ->  
Vec<i32> {  
  
}  
}
```

Ruby:

```
# @param {Integer[][]} flowers  
# @param {Integer[]} people  
# @return {Integer[]}  
def full_bloom_flowers(flowers, people)  
  
end
```

PHP:

```
class Solution {  
  
/**  
 * @param Integer[][] $flowers  
 * @param Integer[] $people  
 * @return Integer[]  
 */  
function fullBloomFlowers($flowers, $people) {  
  
}  
}
```

Dart:

```
class Solution {  
List<int> fullBloomFlowers(List<List<int>> flowers, List<int> people) {
```

```
}
```

```
}
```

Scala:

```
object Solution {  
    def fullBloomFlowers(flowers: Array[Array[Int]], people: Array[Int]):  
        Array[Int] = {  
  
    }  
}
```

Elixir:

```
defmodule Solution do  
    @spec full_bloom_flowers([integer], [integer]) :: [integer]  
    def full_bloom_flowers(flowers, people) do  
  
    end  
end
```

Erlang:

```
-spec full_bloom_flowers([[integer]], [integer]) -> [integer].  
full_bloom_flowers(Flowers, People) ->  
.
```

Racket:

```
(define/contract (full-bloom-flowers flowers people)  
  (-> (listof (listof exact-integer?)) (listof exact-integer?) (listof  
    exact-integer?))  
)
```

Solutions

C++ Solution:

```

/*
 * Problem: Number of Flowers in Full Bloom
 * Difficulty: Hard
 * Tags: array, hash, sort, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) for hash map
 */

class Solution {
public:
vector<int> fullBloomFlowers(vector<vector<int>>& flowers, vector<int>&
people) {

}
};


```

Java Solution:

```

/**
 * Problem: Number of Flowers in Full Bloom
 * Difficulty: Hard
 * Tags: array, hash, sort, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) for hash map
 */

class Solution {
public int[] fullBloomFlowers(int[][] flowers, int[] people) {

}
}


```

Python3 Solution:

```

"""
Problem: Number of Flowers in Full Bloom
Difficulty: Hard
Tags: array, hash, sort, search

```

```

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:

def fullBloomFlowers(self, flowers: List[List[int]], people: List[int]) ->
List[int]:
# TODO: Implement optimized solution
pass

```

Python Solution:

```

class Solution(object):
    def fullBloomFlowers(self, flowers, people):
        """
        :type flowers: List[List[int]]
        :type people: List[int]
        :rtype: List[int]
        """

```

JavaScript Solution:

```

/**
 * Problem: Number of Flowers in Full Bloom
 * Difficulty: Hard
 * Tags: array, hash, sort, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) for hash map
 */

/**
 * @param {number[][]} flowers
 * @param {number[]} people
 * @return {number[]}
 */
var fullBloomFlowers = function(flowers, people) {

```

```
};
```

TypeScript Solution:

```
/**  
 * Problem: Number of Flowers in Full Bloom  
 * Difficulty: Hard  
 * Tags: array, hash, sort, search  
 *  
 * Approach: Use two pointers or sliding window technique  
 * Time Complexity: O(n) or O(n log n)  
 * Space Complexity: O(n) for hash map  
 */  
  
function fullBloomFlowers(flowers: number[][], people: number[]): number[] {  
  
};
```

C# Solution:

```
/*  
 * Problem: Number of Flowers in Full Bloom  
 * Difficulty: Hard  
 * Tags: array, hash, sort, search  
 *  
 * Approach: Use two pointers or sliding window technique  
 * Time Complexity: O(n) or O(n log n)  
 * Space Complexity: O(n) for hash map  
 */  
  
public class Solution {  
    public int[] FullBloomFlowers(int[][] flowers, int[] people) {  
  
    }  
}
```

C Solution:

```
/*  
 * Problem: Number of Flowers in Full Bloom  
 * Difficulty: Hard
```

```

* Tags: array, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/
/***
* Note: The returned array must be malloced, assume caller calls free().
*/
int* fullBloomFlowers(int** flowers, int flowersSize, int* flowersColSize,
int* people, int peopleSize, int* returnSize) {

}

```

Go Solution:

```

// Problem: Number of Flowers in Full Bloom
// Difficulty: Hard
// Tags: array, hash, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func fullBloomFlowers(flowers [][]int, people []int) []int {
}

```

Kotlin Solution:

```

class Solution {
    fun fullBloomFlowers(flowers: Array<IntArray>, people: IntArray): IntArray {
        }
    }
}

```

Swift Solution:

```

class Solution {
    func fullBloomFlowers(_ flowers: [[Int]], _ people: [Int]) -> [Int] {
}

```

```
}
```

```
}
```

Rust Solution:

```
// Problem: Number of Flowers in Full Bloom
// Difficulty: Hard
// Tags: array, hash, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
    pub fn full_bloom_flowers(flowers: Vec<Vec<i32>>, people: Vec<i32>) ->
        Vec<i32> {
        }

    }
}
```

Ruby Solution:

```
# @param {Integer[][]} flowers
# @param {Integer[]} people
# @return {Integer[]}
def full_bloom_flowers(flowers, people)

end
```

PHP Solution:

```
class Solution {

    /**
     * @param Integer[][] $flowers
     * @param Integer[] $people
     * @return Integer[]
     */
    function fullBloomFlowers($flowers, $people) {
```

```
}
```

```
}
```

Dart Solution:

```
class Solution {  
List<int> fullBloomFlowers(List<List<int>> flowers, List<int> people) {  
  
}  
}
```

Scala Solution:

```
object Solution {  
def fullBloomFlowers(flowers: Array[Array[Int]], people: Array[Int]):  
  Array[Int] = {  
  
}  
}
```

Elixir Solution:

```
defmodule Solution do  
@spec full_bloom_flowers([integer], [integer]) :: [integer]  
def full_bloom_flowers(flowers, people) do  
  
end  
end
```

Erlang Solution:

```
-spec full_bloom_flowers([[integer]], [integer]) -> [integer].  
full_bloom_flowers(Flowers, People) ->  
.
```

Racket Solution:

```
(define/contract (full-bloom-flowers flowers people)  
  (-> (listof (listof exact-integer?)) (listof exact-integer?) (listof
```

```
exact-integer? ) )
```

```
)
```