Problem 2570: Merge Two 2D Arrays by Summing Values

Problem Information

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

Problem Description

You are given two

2D

integer arrays

nums1

and

nums2.

nums1[i] = [id

i

, val

i

]

indicate that the number with the id

id

i

has a value equal to

val

i

.

nums2[i] = [id

i

, val

i

]

indicate that the number with the id

id

i

has a value equal to

val

i

.

Each array contains

unique

ids and is sorted in

ascending

order by id.

Merge the two arrays into one array that is sorted in ascending order by id, respecting the following conditions:

Only ids that appear in at least one of the two arrays should be included in the resulting array.

Each id should be included

only once

and its value should be the sum of the values of this id in the two arrays. If the id does not exist in one of the two arrays, then assume its value in that array to be

0

.

Return

the resulting array

. The returned array must be sorted in ascending order by id.

Example 1:

Input:

nums1 = [[1,2],[2,3],[4,5]], nums2 = [[1,4],[3,2],[4,1]]

Output:

[[1,6],[2,3],[3,2],[4,6]]

Explanation:

The resulting array contains the following: - id = 1, the value of this id is 2 + 4 = 6. - id = 2, the value of this id is 3. - id = 3, the value of this id is 2. - id = 4, the value of this id is 5 + 1 = 6.

Example 2:

Input:

nums1 = [[2,4],[3,6],[5,5]], nums2 = [[1,3],[4,3]]

Output:

[[1,3],[2,4],[3,6],[4,3],[5,5]]

Explanation:

There are no common ids, so we just include each id with its value in the resulting list.

Constraints:

1 <= nums1.length, nums2.length <= 200

nums1[i].length == nums2[j].length == 2

1 <= id

i

, val

i

<= 1000

Both arrays contain unique ids.

Both arrays are in strictly ascending order by id.

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> mergeArrays(vector<vector<int>>& nums1,
vector<vector<int>>& nums2) {

}
};
```

Java:

```
class Solution {
public int[][] mergeArrays(int[][] nums1, int[][] nums2) {

}
}
```

Python3:

```
class Solution:
def mergeArrays(self, nums1: List[List[int]], nums2: List[List[int]]) ->
List[List[int]]:
```

Python:

```
class Solution(object):
def mergeArrays(self, nums1, nums2):
"""
:type nums1: List[List[int]]
:type nums2: List[List[int]]
:rtype: List[List[int]]
"""
```

JavaScript:

```
/**
* @param {number[][]} nums1
* @param {number[][]} nums2
* @return {number[][]}
*/
```

```
var mergeArrays = function(nums1, nums2) {

};
```

TypeScript:

```
function mergeArrays(nums1: number[][], nums2: number[][]): number[][] {

};
```

C#:

```
public class Solution {
public int[][] MergeArrays(int[][] nums1, int[][] nums2) {

}
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** mergeArrays(int** nums1, int nums1Size, int* nums1ColSize, int** nums2,
int nums2Size, int* nums2ColSize, int* returnSize, int** returnColumnSizes) {

}
```

Go:

```
func mergeArrays(nums1 [][]int, nums2 [][]int) [][]int {

}
```

Kotlin:

```
class Solution {
fun mergeArrays(nums1: Array<IntArray>, nums2: Array<IntArray>):
Array<IntArray> {
```

```
}
}
```

Swift:

```
class Solution {
func mergeArrays(_ nums1: [[Int]], _ nums2: [[Int]]) -> [[Int]] {

}
}
```

Rust:

```
impl Solution {
pub fn merge_arrays(nums1: Vec<Vec<i32>>, nums2: Vec<Vec<i32>>) ->
Vec<Vec<i32>> {

}
}
```

Ruby:

```
# @param {Integer[][]} nums1
# @param {Integer[][]} nums2
# @return {Integer[][]}
def merge_arrays(nums1, nums2)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $nums1
* @param Integer[][] $nums2
* @return Integer[][]
*/
function mergeArrays($nums1, $nums2) {

}
```

```
}
```

Dart:

```
class Solution {
List<List<int>> mergeArrays(List<List<int>> nums1, List<List<int>> nums2) {

}
}
```

Scala:

```
object Solution {
def mergeArrays(nums1: Array[Array[Int]], nums2: Array[Array[Int]]):
Array[Array[Int]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec merge_arrays(nums1 :: [[integer]], nums2 :: [[integer]]) :: [[integer]]
def merge_arrays(nums1, nums2) do

end
end
```

Erlang:

```
-spec merge_arrays(Nums1 :: [[integer()]], Nums2 :: [[integer()]]) ->
[[integer()]].
merge_arrays(Nums1, Nums2) ->
.
```

Racket:

```
(define/contract (merge-arrays nums1 nums2)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)) (listof
(listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Merge Two 2D Arrays by Summing Values
* Difficulty: Easy
* Tags: array, hash, sort
*
* 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<vector<int>> mergeArrays(vector<vector<int>>& nums1,
vector<vector<int>>& nums2) {

}
};
```

Java Solution:

```
/**
* Problem: Merge Two 2D Arrays by Summing Values
* Difficulty: Easy
* Tags: array, hash, sort
*
* 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[][] mergeArrays(int[][] nums1, int[][] nums2) {

}
}
```

Python3 Solution:

```
"""
Problem: Merge Two 2D Arrays by Summing Values
Difficulty: Easy
Tags: array, hash, sort

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 mergeArrays(self, nums1: List[List[int]], nums2: List[List[int]]) ->
List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def mergeArrays(self, nums1, nums2):
"""
:type nums1: List[List[int]]
:type nums2: List[List[int]]
:rtype: List[List[int]]
"""
```

JavaScript Solution:

```
/**
* Problem: Merge Two 2D Arrays by Summing Values
* Difficulty: Easy
* Tags: array, hash, sort
*
* 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[][]} nums1
* @param {number[][]} nums2
* @return {number[][]}
```

```
*/
var mergeArrays = function(nums1, nums2) {

};
```

TypeScript Solution:

```
/**
* Problem: Merge Two 2D Arrays by Summing Values
* Difficulty: Easy
* Tags: array, hash, sort
*
* 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 mergeArrays(nums1: number[][], nums2: number[][]): number[][] {

};
```

C# Solution:

```
/*
* Problem: Merge Two 2D Arrays by Summing Values
* Difficulty: Easy
* Tags: array, hash, sort
*
* 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[][] MergeArrays(int[][] nums1, int[][] nums2) {

}
}
```

C Solution:

```
/*
* Problem: Merge Two 2D Arrays by Summing Values
* Difficulty: Easy
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** mergeArrays(int** nums1, int nums1Size, int* nums1ColSize, int** nums2,
int nums2Size, int* nums2ColSize, int* returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Merge Two 2D Arrays by Summing Values
// Difficulty: Easy
// Tags: array, hash, sort
//
// 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 mergeArrays(nums1 [][]int, nums2 [][]int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun mergeArrays(nums1: Array<IntArray>, nums2: Array<IntArray>):
Array<IntArray> {

}
```

```
}
```

Swift Solution:

```
class Solution {
func mergeArrays(_ nums1: [[Int]], _ nums2: [[Int]]) -> [[Int]] {

}
}
```

Rust Solution:

```
// Problem: Merge Two 2D Arrays by Summing Values
// Difficulty: Easy
// Tags: array, hash, sort
//
// 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 merge_arrays(nums1: Vec<Vec<i32>>, nums2: Vec<Vec<i32>>) ->
Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} nums1
# @param {Integer[][]} nums2
# @return {Integer[][]}
def merge_arrays(nums1, nums2)

end
```

PHP Solution:

```
class Solution {

/**
```

```
* @param Integer[][] $nums1
* @param Integer[][] $nums2
* @return Integer[][]
*/
function mergeArrays($nums1, $nums2) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> mergeArrays(List<List<int>> nums1, List<List<int>> nums2) {

}
}
```

Scala Solution:

```
object Solution {
def mergeArrays(nums1: Array[Array[Int]], nums2: Array[Array[Int]]):
Array[Array[Int]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec merge_arrays(nums1 :: [[integer]], nums2 :: [[integer]]) :: [[integer]]
def merge_arrays(nums1, nums2) do

end
end
```

Erlang Solution:

```
-spec merge_arrays(Nums1 :: [[integer()]], Nums2 :: [[integer()]]) ->
[[integer()]].
merge_arrays(Nums1, Nums2) ->
.
```

Racket Solution:

```
(define/contract (merge-arrays nums1 nums2)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)) (listof
(listof exact-integer?)))
)
```