Problem 2363: Merge Similar Items

Problem Information

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

Problem Description

You are given two 2D integer arrays,

items1

and

items2

, representing two sets of items. Each array

items

has the following properties:

items[i] = [value

i

, weight

i

]

where

value

i

represents the

value

and

weight

i

represents the

weight

of the

i

th

item.

The value of each item in

items

is

unique

.

Return

a 2D integer array

ret

where

ret[i] = [value

i

, weight

i

]

,

with

weight

i

being the

sum of weights

of all items with value

value

i

.

Note:

ret

should be returned in

ascending

order by value.

Example 1:

Input:

items1 = [[1,1],[4,5],[3,8]], items2 = [[3,1],[1,5]]

Output:

[[1,6],[3,9],[4,5]]

Explanation:

The item with value = 1 occurs in items1 with weight = 1 and in items2 with weight = 5, total weight = 1 + 5 = 6. The item with value = 3 occurs in items1 with weight = 8 and in items2 with weight = 1, total weight = 8 + 1 = 9. The item with value = 4 occurs in items1 with weight = 5, total weight = 5. Therefore, we return [[1,6],[3,9],[4,5]].

Example 2:

Input:

items1 = [[1,1],[3,2],[2,3]], items2 = [[2,1],[3,2],[1,3]]

Output:

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

Explanation:

The item with value = 1 occurs in items1 with weight = 1 and in items2 with weight = 3, total weight = 1 + 3 = 4. The item with value = 2 occurs in items1 with weight = 3 and in items2 with weight = 1, total weight = 3 + 1 = 4. The item with value = 3 occurs in items1 with weight = 2 and in items2 with weight = 2, total weight = 2 + 2 = 4. Therefore, we return [[1,4],[2,4],[3,4]].

Example 3:

Input:

items1 = [[1,3],[2,2]], items2 = [[7,1],[2,2],[1,4]]

Output:

[[1,7],[2,4],[7,1]]

Explanation:

The item with value = 1 occurs in items1 with weight = 3 and in items2 with weight = 4, total weight = 3 + 4 = 7. The item with value = 2 occurs in items1 with weight = 2 and in items2 with weight = 2, total weight = 2 + 2 = 4. The item with value = 7 occurs in items2 with weight = 1, total weight = 1. Therefore, we return [[1,7],[2,4],[7,1]].

Constraints:

1 <= items1.length, items2.length <= 1000

items1[i].length == items2[i].length == 2

1 <= value

i

, weight

i

<= 1000

Each

value

i

in

items1

is

unique

.

Each

value

i

in

items2

is

unique

.

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> mergeSimilarItems(vector<vector<int>>& items1,
vector<vector<int>>& items2) {

}
};
```

Java:

```
class Solution {
public List<List<Integer>> mergeSimilarItems(int[][] items1, int[][] items2)
{

}
}
```

Python3:

```
class Solution:
def mergeSimilarItems(self, items1: List[List[int]], items2: List[List[int]])
-> List[List[int]]:
```

Python:

```
class Solution(object):
def mergeSimilarItems(self, items1, items2):
"""
:type items1: List[List[int]]
:type items2: List[List[int]]
:rtype: List[List[int]]
"""
```

JavaScript:

```
/**
* @param {number[][]} items1
* @param {number[][]} items2
* @return {number[][]}
*/
var mergeSimilarItems = function(items1, items2) {

};
```

TypeScript:

```
function mergeSimilarItems(items1: number[][], items2: number[][]):
number[][] {

};
```

C#:

```
public class Solution {
public IList<IList<int>> MergeSimilarItems(int[][] items1, int[][] items2) {

}
}
```

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** mergeSimilarItems(int** items1, int items1Size, int* items1ColSize,
int** items2, int items2Size, int* items2ColSize, int* returnSize, int**
returnColumnSizes) {

}
```

Go:

```
func mergeSimilarItems(items1 [][]int, items2 [][]int) [][]int {

}
```

Kotlin:

```
class Solution {
fun mergeSimilarItems(items1: Array<IntArray>, items2: Array<IntArray>):
List<List<Int>> {

}
}
```

Swift:

```
class Solution {
func mergeSimilarItems(_ items1: [[Int]], _ items2: [[Int]]) -> [[Int]] {

}
}
```

Rust:

```
impl Solution {
pub fn merge_similar_items(items1: Vec<Vec<i32>>, items2: Vec<Vec<i32>>) ->
Vec<Vec<i32>> {

}
}
```

Ruby:

```
# @param {Integer[][]} items1
# @param {Integer[][]} items2
# @return {Integer[][]}
def merge_similar_items(items1, items2)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $items1
* @param Integer[][] $items2
* @return Integer[][]
*/
function mergeSimilarItems($items1, $items2) {

}
}
```

Dart:

```
class Solution {
List<List<int>> mergeSimilarItems(List<List<int>> items1, List<List<int>>
items2) {

}
}
```

Scala:

```
object Solution {
def mergeSimilarItems(items1: Array[Array[Int]], items2: Array[Array[Int]]):
List[List[Int]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec merge_similar_items(items1 :: [[integer]], items2 :: [[integer]]) ::
[[integer]]
def merge_similar_items(items1, items2) do

end
end
```

Erlang:

```
-spec merge_similar_items(Items1 :: [[integer()]], Items2 :: [[integer()]])
-> [[integer()]].
merge_similar_items(Items1, Items2) ->
.
```

Racket:

```
(define/contract (merge-similar-items items1 items2)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)) (listof
(listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Merge Similar Items
* 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>> mergeSimilarItems(vector<vector<int>>& items1,
vector<vector<int>>& items2) {

}
};
```

Java Solution:

```
/**
* Problem: Merge Similar Items
* 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 List<List<Integer>> mergeSimilarItems(int[][] items1, int[][] items2)
{

}
}
```

Python3 Solution:

```
"""
Problem: Merge Similar Items
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 mergeSimilarItems(self, items1: List[List[int]], items2: List[List[int]])
-> List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def mergeSimilarItems(self, items1, items2):
"""
:type items1: List[List[int]]
:type items2: List[List[int]]
:rtype: List[List[int]]
"""
```

JavaScript Solution:

```
/**
* Problem: Merge Similar Items
* 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[][]} items1
* @param {number[][]} items2
* @return {number[][]}
*/
var mergeSimilarItems = function(items1, items2) {

};
```

TypeScript Solution:

```
/**
* Problem: Merge Similar Items
* 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 mergeSimilarItems(items1: number[][], items2: number[][]):
number[][] {

};
```

C# Solution:

```
/*
* Problem: Merge Similar Items
* 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 IList<IList<int>> MergeSimilarItems(int[][] items1, int[][] items2) {

}
}
```

C Solution:

```
/*
* Problem: Merge Similar Items
* 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** mergeSimilarItems(int** items1, int items1Size, int* items1ColSize,
int** items2, int items2Size, int* items2ColSize, int* returnSize, int**
returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Merge Similar Items
// 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 mergeSimilarItems(items1 [][]int, items2 [][]int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun mergeSimilarItems(items1: Array<IntArray>, items2: Array<IntArray>):
List<List<Int>> {

}
}
```

Swift Solution:

```
class Solution {
func mergeSimilarItems(_ items1: [[Int]], _ items2: [[Int]]) -> [[Int]] {

}
}
```

Rust Solution:

```
// Problem: Merge Similar Items
// 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_similar_items(items1: Vec<Vec<i32>>, items2: Vec<Vec<i32>>) ->
Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} items1
# @param {Integer[][]} items2
# @return {Integer[][]}
def merge_similar_items(items1, items2)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $items1
* @param Integer[][] $items2
* @return Integer[][]
*/
function mergeSimilarItems($items1, $items2) {



}
}
```

Dart Solution:

```
class Solution {
List<List<int>> mergeSimilarItems(List<List<int>> items1, List<List<int>>
items2) {

}
}
```

Scala Solution:

```
object Solution {
def mergeSimilarItems(items1: Array[Array[Int]], items2: Array[Array[Int]]):
List[List[Int]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec merge_similar_items(items1 :: [[integer]], items2 :: [[integer]]) ::
[[integer]]
def merge_similar_items(items1, items2) do

end
end
```

Erlang Solution:

```
-spec merge_similar_items(Items1 :: [[integer()]], Items2 :: [[integer()]])
-> [[integer()]].
merge_similar_items(Items1, Items2) ->
.
```

Racket Solution:

```
(define/contract (merge-similar-items items1 items2)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)) (listof
(listof exact-integer?)))
)
```