Problem 1605: Find Valid Matrix Given Row and Column Sums

Problem Information

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

Problem Description

You are given two arrays

rowSum

and

colSum

of non-negative integers where

rowSum[i]

is the sum of the elements in the

i

th

row and

colSum[j]

is the sum of the elements of the

j

th

column of a 2D matrix. In other words, you do not know the elements of the matrix, but you do know the sums of each row and column.

Find any matrix of

non-negative

integers of size

rowSum.length x colSum.length

that satisfies the

rowSum

and

colSum

requirements.

Return

a 2D array representing

any

matrix that fulfills the requirements

. It's guaranteed that

at least one

matrix that fulfills the requirements exists.

Example 1:

Input:

rowSum = [3,8], colSum = [4,7]

Output:

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

Explanation:

0

th

row: 3 + 0 = 3 == rowSum[0] 1

st

row: 1 + 7 = 8 == rowSum[1] 0

th

column: 3 + 1 = 4 == colSum[0] 1

st

column: 0 + 7 = 7 == colSum[1] The row and column sums match, and all matrix elements are non-negative. Another possible matrix is: [[1,2], [3,5]]

Example 2:

Input:

rowSum = [5,7,10], colSum = [8,6,8]

Output:

[[0,5,0], [6,1,0], [2,0,8]]

Constraints:

1 <= rowSum.length, colSum.length <= 500

0 <= rowSum[i], colSum[i] <= 10

8

sum(rowSum) == sum(colSum)

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> restoreMatrix(vector<int>& rowSum, vector<int>& colSum) {

}
};
```

Java:

```
class Solution {
public int[][] restoreMatrix(int[] rowSum, int[] colSum) {

}
}
```

Python3:

```
class Solution:
def restoreMatrix(self, rowSum: List[int], colSum: List[int]) ->
List[List[int]]:
```

Python:

```
class Solution(object):
def restoreMatrix(self, rowSum, colSum):
"""
```

```
:type rowSum: List[int]
:type colSum: List[int]
:rtype: List[List[int]]
"""
```

JavaScript:

```
/**
* @param {number[]} rowSum
* @param {number[]} colSum
* @return {number[][]}
*/
var restoreMatrix = function(rowSum, colSum) {

};
```

TypeScript:

```
function restoreMatrix(rowSum: number[], colSum: number[]): number[][] {

};
```

C#:

```
public class Solution {
public int[][] RestoreMatrix(int[] rowSum, int[] colSum) {

}
}
```

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** restoreMatrix(int* rowSum, int rowSumSize, int* colSum, int colSumSize,
int* returnSize, int** returnColumnSizes) {

}
```

Go:

```
func restoreMatrix(rowSum []int, colSum []int) [][]int {

}
```

Kotlin:

```
class Solution {
fun restoreMatrix(rowSum: IntArray, colSum: IntArray): Array<IntArray> {

}
}
```

Swift:

```
class Solution {
func restoreMatrix(_ rowSum: [Int], _ colSum: [Int]) -> [[Int]] {

}
}
```

Rust:

```
impl Solution {
pub fn restore_matrix(row_sum: Vec<i32>, col_sum: Vec<i32>) -> Vec<Vec<i32>>
{

}
}
```

Ruby:

```
# @param {Integer[]} row_sum
# @param {Integer[]} col_sum
# @return {Integer[][]}
def restore_matrix(row_sum, col_sum)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $rowSum
* @param Integer[] $colSum
* @return Integer[][]
*/
function restoreMatrix($rowSum, $colSum) {

}
}
```

Dart:

```
class Solution {
List<List<int>> restoreMatrix(List<int> rowSum, List<int> colSum) {

}
}
```

Scala:

```
object Solution {
def restoreMatrix(rowSum: Array[Int], colSum: Array[Int]): Array[Array[Int]]
= {

}
}
```

Elixir:

```
defmodule Solution do
@spec restore_matrix(row_sum :: [integer], col_sum :: [integer]) ::
[[integer]]
def restore_matrix(row_sum, col_sum) do

end
end
```

Erlang:

```
-spec restore_matrix(RowSum :: [integer()], ColSum :: [integer()]) ->
[[integer()]].
```

```
restore_matrix(RowSum, ColSum) ->
.
```

Racket:

```
(define/contract (restore-matrix rowSum colSum)
(-> (listof exact-integer?) (listof exact-integer?) (listof (listof
exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find Valid Matrix Given Row and Column Sums
* Difficulty: Medium
* Tags: array, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
vector<vector<int>> restoreMatrix(vector<int>& rowSum, vector<int>& colSum) {

}
};
```

Java Solution:

```
/**
* Problem: Find Valid Matrix Given Row and Column Sums
* Difficulty: Medium
* Tags: array, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int[][] restoreMatrix(int[] rowSum, int[] colSum) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Valid Matrix Given Row and Column Sums
Difficulty: Medium
Tags: array, greedy

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def restoreMatrix(self, rowSum: List[int], colSum: List[int]) ->
List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def restoreMatrix(self, rowSum, colSum):
"""
:type rowSum: List[int]
:type colSum: List[int]
:rtype: List[List[int]]
"""
```

JavaScript Solution:

```
/**
* Problem: Find Valid Matrix Given Row and Column Sums
```

```
* Difficulty: Medium
* Tags: array, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number[]} rowSum
* @param {number[]} colSum
* @return {number[][]}
*/
var restoreMatrix = function(rowSum, colSum) {

};
```

TypeScript Solution:

```
/**
* Problem: Find Valid Matrix Given Row and Column Sums
* Difficulty: Medium
* Tags: array, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function restoreMatrix(rowSum: number[], colSum: number[]): number[][] {

};
```

C# Solution:

```
/*
* Problem: Find Valid Matrix Given Row and Column Sums
* Difficulty: Medium
* Tags: array, greedy
*
* Approach: Use two pointers or sliding window technique
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int[][] RestoreMatrix(int[] rowSum, int[] colSum) {

}
}
```

C Solution:

```
/*
* Problem: Find Valid Matrix Given Row and Column Sums
* Difficulty: Medium
* Tags: array, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* 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** restoreMatrix(int* rowSum, int rowSumSize, int* colSum, int colSumSize,
int* returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Find Valid Matrix Given Row and Column Sums
// Difficulty: Medium
// Tags: array, greedy
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
```

```
// Space Complexity: O(1) to O(n) depending on approach

func restoreMatrix(rowSum []int, colSum []int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun restoreMatrix(rowSum: IntArray, colSum: IntArray): Array<IntArray> {

}
}
```

Swift Solution:

```
class Solution {
func restoreMatrix(_ rowSum: [Int], _ colSum: [Int]) -> [[Int]] {

}
}
```

Rust Solution:

```
// Problem: Find Valid Matrix Given Row and Column Sums
// Difficulty: Medium
// Tags: array, greedy
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn restore_matrix(row_sum: Vec<i32>, col_sum: Vec<i32>) -> Vec<Vec<i32>>
{

}
}
```

Ruby Solution:

```
# @param {Integer[]} row_sum
# @param {Integer[]} col_sum
# @return {Integer[][]}
def restore_matrix(row_sum, col_sum)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $rowSum
* @param Integer[] $colSum
* @return Integer[][]
*/
function restoreMatrix($rowSum, $colSum) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> restoreMatrix(List<int> rowSum, List<int> colSum) {

}
}
```

Scala Solution:

```
object Solution {
def restoreMatrix(rowSum: Array[Int], colSum: Array[Int]): Array[Array[Int]]
= {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec restore_matrix(row_sum :: [integer], col_sum :: [integer]) ::
```

```
[[integer]]
def restore_matrix(row_sum, col_sum) do

end
end
```

Erlang Solution:

```
-spec restore_matrix(RowSum :: [integer()], ColSum :: [integer()]) ->
[[integer()]].
restore_matrix(RowSum, ColSum) ->
.
```

Racket Solution:

```
(define/contract (restore-matrix rowSum colSum)
(-> (listof exact-integer?) (listof exact-integer?) (listof (listof
exact-integer?)))
)
```