Problem 632: Smallest Range Covering Elements from K Lists

Problem Information

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

Problem Description

You have

k

lists of sorted integers in

non-decreasing order

. Find the

smallest

range that includes at least one number from each of the

k

lists.

We define the range

[a, b]

is smaller than range

[c, d]

if

$b - a < d - c$

or

$a < c$

if

$b - a == d - c$

.

Example 1:

Input:

nums = [[4,10,15,24,26],[0,9,12,20],[5,18,22,30]]

Output:

[20,24]

Explanation:

List 1: [4, 10, 15, 24,26], 24 is in range [20,24]. List 2: [0, 9, 12, 20], 20 is in range [20,24]. List 3: [5, 18, 22, 30], 22 is in range [20,24].

Example 2:

Input:

nums = [[1,2,3],[1,2,3],[1,2,3]]

Output:

[1,1]

Constraints:

nums.length == k

1 <= k <= 3500

1 <= nums[i].length <= 50

-10

5

<= nums[i][j] <= 10

5

nums[i]

is sorted in

non-decreasing

order.

Code Snippets

C++:

```
class Solution {
public:
vector<int> smallestRange(vector<vector<int>>& nums) {

}
};
```

Java:

```
class Solution {
public int[] smallestRange(List<List<Integer>> nums) {
}
}
```

Python3:

```
class Solution:
def smallestRange(self, nums: List[List[int]]) -> List[int]:
```

Python:

```
class Solution(object):
def smallestRange(self, nums):
"""
:type nums: List[List[int]]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[][]} nums
* @return {number[]}
*/
var smallestRange = function(nums) {

};
```

TypeScript:

```
function smallestRange(nums: number[][]): number[] {

};
```

C#:

```
public class Solution {
public int[] SmallestRange(IList<IList<int>> nums) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* smallestRange(int** nums, int numsSize, int* numsColSize, int*
returnSize) {

}
```

Go:

```
func smallestRange(nums [][]int) []int {

}
```

Kotlin:

```
class Solution {
fun smallestRange(nums: List<List<Int>>): IntArray {

}
}
```

Swift:

```
class Solution {
func smallestRange(_ nums: [[Int]]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn smallest_range(nums: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[][]} nums
# @return {Integer[]}
def smallest_range(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $nums
* @return Integer[]
*/
function smallestRange($nums) {

}
}
```

Dart:

```
class Solution {
List<int> smallestRange(List<List<int>> nums) {

}
}
```

Scala:

```
object Solution {
def smallestRange(nums: List[List[Int]]): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec smallest_range(nums :: [[integer]]) :: [integer]
def smallest_range(nums) do

end
end
```

Erlang:

```
-spec smallest_range(Nums :: [[integer()]]) -> [integer()].
smallest_range(Nums) ->
.
```

Racket:

```
(define/contract (smallest-range nums)
(-> (listof (listof exact-integer?)) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Smallest Range Covering Elements from K Lists
* Difficulty: Hard
* Tags: array, greedy, hash, sort, queue, heap
*
* 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> smallestRange(vector<vector<int>>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Smallest Range Covering Elements from K Lists
* Difficulty: Hard
* Tags: array, greedy, hash, sort, queue, heap
*
* 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[] smallestRange(List<List<Integer>> nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Smallest Range Covering Elements from K Lists
Difficulty: Hard
Tags: array, greedy, hash, sort, queue, heap

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 smallestRange(self, nums: List[List[int]]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def smallestRange(self, nums):
"""
:type nums: List[List[int]]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Smallest Range Covering Elements from K Lists
* Difficulty: Hard
* Tags: array, greedy, hash, sort, queue, heap
*
* 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[][]} nums
* @return {number[]}
*/
var smallestRange = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Smallest Range Covering Elements from K Lists
* Difficulty: Hard
* Tags: array, greedy, hash, sort, queue, heap
*
* 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 smallestRange(nums: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Smallest Range Covering Elements from K Lists
* Difficulty: Hard
* Tags: array, greedy, hash, sort, queue, heap
*
* 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[] SmallestRange(IList<IList<int>> nums) {

}
}
```

C Solution:

```
/*
* Problem: Smallest Range Covering Elements from K Lists
* Difficulty: Hard
* Tags: array, greedy, hash, sort, queue, heap
*
* 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* smallestRange(int** nums, int numsSize, int* numsColSize, int*
returnSize) {

}
```

Go Solution:

```
// Problem: Smallest Range Covering Elements from K Lists
// Difficulty: Hard
// Tags: array, greedy, hash, sort, queue, heap
//
// 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 smallestRange(nums [][]int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun smallestRange(nums: List<List<Int>>): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func smallestRange(_ nums: [[Int]]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Smallest Range Covering Elements from K Lists
// Difficulty: Hard
// Tags: array, greedy, hash, sort, queue, heap
//
// 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 smallest_range(nums: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} nums
# @return {Integer[]}
def smallest_range(nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $nums
* @return Integer[]
*/
function smallestRange($nums) {

}
}
```

Dart Solution:

```
class Solution {
List<int> smallestRange(List<List<int>> nums) {

}
}
```

Scala Solution:

```
object Solution {
def smallestRange(nums: List[List[Int]]): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec smallest_range(nums :: [[integer]]) :: [integer]
def smallest_range(nums) do

end
end
```

Erlang Solution:

```
-spec smallest_range(Nums :: [[integer()]]) -> [integer()].
smallest_range(Nums) ->
.
```

Racket Solution:

```
(define/contract (smallest-range nums)
(-> (listof (listof exact-integer?)) (listof exact-integer?))
)
```