Problem 1940: Longest Common Subsequence Between Sorted Arrays

Problem Information

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

Problem Description

Given an array of integer arrays

arrays

where each

arrays[i]

is sorted in

strictly increasing

order, return

an integer array representing the

longest common subsequence

among

all

the arrays

.

A

subsequence

is a sequence that can be derived from another sequence by deleting some elements (possibly none) without changing the order of the remaining elements.

Example 1:

Input:

arrays = [[

1

,3,

4

], [

1

,

4

,7,9]]

Output:

[1,4]

Explanation:

The longest common subsequence in the two arrays is [1,4].

Example 2:

Input:

arrays = [[

2

,

3

,

6

,8], [1,

2

,

3

,5,

6

,7,10], [

2

,

3

,4,

6

,9]]

Output:

[2,3,6]

Explanation:

The longest common subsequence in all three arrays is [2,3,6].

Example 3:

Input:

arrays = [[1,2,3,4,5], [6,7,8]]

Output:

[]

Explanation:

There is no common subsequence between the two arrays.

Constraints:

2 <= arrays.length <= 100

1 <= arrays[i].length <= 100

1 <= arrays[i][j] <= 100

arrays[i]

is sorted in

strictly increasing

order.

Code Snippets

C++:

```
class Solution {
public:
vector<int> longestCommonSubsequence(vector<vector<int>>& arrays) {

}
};
```

Java:

```
class Solution {
public List<Integer> longestCommonSubsequence(int[][] arrays) {

}
}
```

Python3:

```
class Solution:
def longestCommonSubsequence(self, arrays: List[List[int]]) -> List[int]:
```

Python:

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

JavaScript:

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

```
};
```

TypeScript:

```
function longestCommonSubsequence(arrays: number[][]): number[] {

};
```

C#:

```
public class Solution {
public IList<int> LongestCommonSubsequence(int[][] arrays) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* longestCommonSubsequence(int** arrays, int arraysSize, int*
arraysColSize, int* returnSize) {

}
```

Go:

```
func longestCommonSubsequence(arrays [][]int) []int {

}
```

Kotlin:

```
class Solution {
fun longestCommonSubsequence(arrays: Array<IntArray>): List<Int> {

}
}
```

Swift:

```
class Solution {
func longestCommonSubsequence(_ arrays: [[Int]]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn longest_common_subsequence(arrays: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[][]} arrays
# @return {Integer[]}
def longest_common_subsequence(arrays)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $arrays
* @return Integer[]
*/
function longestCommonSubsequence($arrays) {

}
}
```

Dart:

```
class Solution {
List<int> longestCommonSubsequence(List<List<int>> arrays) {

}
}
```

Scala:

```
object Solution {
def longestCommonSubsequence(arrays: Array[Array[Int]]): List[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec longest_common_subsequence(arrays :: [[integer]]) :: [integer]
def longest_common_subsequence(arrays) do

end
end
```

Erlang:

```
-spec longest_common_subsequence(Arrays :: [[integer()]]) -> [integer()].
longest_common_subsequence(Arrays) ->
.
```

Racket:

```
(define/contract (longest-common-subsequence arrays)
(-> (listof (listof exact-integer?)) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Longest Common Subsequence Between Sorted Arrays
* Difficulty: Medium
* 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<int> longestCommonSubsequence(vector<vector<int>>& arrays) {

}
};
```

Java Solution:

```
/**
* Problem: Longest Common Subsequence Between Sorted Arrays
* Difficulty: Medium
* 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<Integer> longestCommonSubsequence(int[][] arrays) {

}
}
```

Python3 Solution:

```
"""
Problem: Longest Common Subsequence Between Sorted Arrays
Difficulty: Medium
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 longestCommonSubsequence(self, arrays: List[List[int]]) -> List[int]:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Longest Common Subsequence Between Sorted Arrays
* Difficulty: Medium
* 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[][]} arrays
* @return {number[]}
*/
var longestCommonSubsequence = function(arrays) {

};
```

TypeScript Solution:

```
/**
* Problem: Longest Common Subsequence Between Sorted Arrays
* Difficulty: Medium
* 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 longestCommonSubsequence(arrays: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Longest Common Subsequence Between Sorted Arrays
* Difficulty: Medium
* 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<int> LongestCommonSubsequence(int[][] arrays) {

}
}
```

C Solution:

```
/*
* Problem: Longest Common Subsequence Between Sorted Arrays
* Difficulty: Medium
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* longestCommonSubsequence(int** arrays, int arraysSize, int*
arraysColSize, int* returnSize) {
```

```
}
```

Go Solution:

```
// Problem: Longest Common Subsequence Between Sorted Arrays
// Difficulty: Medium
// 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 longestCommonSubsequence(arrays [][]int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun longestCommonSubsequence(arrays: Array<IntArray>): List<Int> {

}
}
```

Swift Solution:

```
class Solution {
func longestCommonSubsequence(_ arrays: [[Int]]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Longest Common Subsequence Between Sorted Arrays
// Difficulty: Medium
// 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 longest_common_subsequence(arrays: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} arrays
# @return {Integer[]}
def longest_common_subsequence(arrays)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $arrays
* @return Integer[]
*/
function longestCommonSubsequence($arrays) {

}
}
```

Dart Solution:

```
class Solution {
List<int> longestCommonSubsequence(List<List<int>> arrays) {

}
}
```

Scala Solution:

```
object Solution {
def longestCommonSubsequence(arrays: Array[Array[Int]]): List[Int] = {
```

```
}
}
```

Elixir Solution:

```
defmodule Solution do
@spec longest_common_subsequence(arrays :: [[integer]]) :: [integer]
def longest_common_subsequence(arrays) do

end
end
```

Erlang Solution:

```
-spec longest_common_subsequence(Arrays :: [[integer()]]) -> [integer()].
longest_common_subsequence(Arrays) ->
.
```

Racket Solution:

```
(define/contract (longest-common-subsequence arrays)
(-> (listof (listof exact-integer?)) (listof exact-integer?))
)
```