Problem 1337: The K Weakest Rows in a Matrix

Problem Information

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

Problem Description

You are given an

m x n

binary matrix

mat

of

1

's (representing soldiers) and

0

's (representing civilians). The soldiers are positioned

in front

of the civilians. That is, all the

1

's will appear to the

left

of all the

0

's in each row.

A row

i

is

weaker

than a row

j

if one of the following is true:

The number of soldiers in row

i

is less than the number of soldiers in row

j

.

Both rows have the same number of soldiers and

i < j

.

Return

the indices of the

k

weakest

rows in the matrix ordered from weakest to strongest

.

Example 1:

Input:

mat = [[1,1,0,0,0], [1,1,1,1,0], [1,0,0,0,0], [1,1,0,0,0], [1,1,1,1,1]], k = 3

Output:

[2,0,3]

Explanation:

The number of soldiers in each row is: - Row 0: 2 - Row 1: 4 - Row 2: 1 - Row 3: 2 - Row 4: 5 The rows ordered from weakest to strongest are [2,0,3,1,4].

Example 2:

Input:

mat = [[1,0,0,0], [1,1,1,1], [1,0,0,0], [1,0,0,0]], k = 2

Output:

[0,2]

Explanation:

The number of soldiers in each row is: - Row 0: 1 - Row 1: 4 - Row 2: 1 - Row 3: 1 The rows ordered from weakest to strongest are [0,2,3,1].

Constraints:

m == mat.length

n == mat[i].length

2 <= n, m <= 100

1 <= k <= m

matrix[i][j]

is either 0 or 1.

Code Snippets

C++:

```
class Solution {
public:
vector<int> kWeakestRows(vector<vector<int>>& mat, int k) {

}
};
```

Java:

```
class Solution {
public int[] kWeakestRows(int[][] mat, int k) {

}
}
```

Python3:

```
class Solution:
def kWeakestRows(self, mat: List[List[int]], k: int) -> List[int]:
```

Python:

```
class Solution(object):
def kWeakestRows(self, mat, k):
"""
:type mat: List[List[int]]
:type k: int
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[][]} mat
* @param {number} k
* @return {number[]}
*/
var kWeakestRows = function(mat, k) {

};
```

TypeScript:

```
function kWeakestRows(mat: number[][], k: number): number[] {

};
```

C#:

```
public class Solution {
public int[] KWeakestRows(int[][] mat, int k) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* kWeakestRows(int** mat, int matSize, int* matColSize, int k, int*
returnSize) {

}
```

Go:

```
func kWeakestRows(mat [][]int, k int) []int {

}
```

Kotlin:

```
class Solution {
fun kWeakestRows(mat: Array<IntArray>, k: Int): IntArray {

}
}
```

Swift:

```
class Solution {
func kWeakestRows(_ mat: [[Int]], _ k: Int) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn k_weakest_rows(mat: Vec<Vec<i32>>, k: i32) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[][]} mat
# @param {Integer} k
# @return {Integer[]}
def k_weakest_rows(mat, k)

end
```

PHP:

```
class Solution {
```

```
/**
* @param Integer[][] $mat
* @param Integer $k
* @return Integer[]
*/
function kWeakestRows($mat, $k) {

}
}
```

Dart:

```
class Solution {
List<int> kWeakestRows(List<List<int>> mat, int k) {

}
}
```

Scala:

```
object Solution {
def kWeakestRows(mat: Array[Array[Int]], k: Int): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec k_weakest_rows(mat :: [[integer]], k :: integer) :: [integer]
def k_weakest_rows(mat, k) do

end
end
```

Erlang:

```
-spec k_weakest_rows(Mat :: [[integer()]], K :: integer()) -> [integer()].
k_weakest_rows(Mat, K) ->
.
```

Racket:

```
(define/contract (k-weakest-rows mat k)
(-> (listof (listof exact-integer?)) exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: The K Weakest Rows in a Matrix
* Difficulty: Easy
* Tags: array, sort, search, queue, heap
*
* 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<int> kWeakestRows(vector<vector<int>>& mat, int k) {

}
};
```

Java Solution:

```
/**
* Problem: The K Weakest Rows in a Matrix
* Difficulty: Easy
* Tags: array, sort, search, queue, heap
*
* 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[] kWeakestRows(int[][] mat, int k) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: The K Weakest Rows in a Matrix
Difficulty: Easy
Tags: array, sort, search, queue, heap

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 kWeakestRows(self, mat: List[List[int]], k: int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def kWeakestRows(self, mat, k):
"""
:type mat: List[List[int]]
:type k: int
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: The K Weakest Rows in a Matrix
* Difficulty: Easy
* Tags: array, sort, search, queue, heap
*
* 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[][]} mat
* @param {number} k
* @return {number[]}
*/
var kWeakestRows = function(mat, k) {

};
```

TypeScript Solution:

```
/**
* Problem: The K Weakest Rows in a Matrix
* Difficulty: Easy
* Tags: array, sort, search, queue, heap
*
* 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 kWeakestRows(mat: number[][], k: number): number[] {

};
```

C# Solution:

```
/*
* Problem: The K Weakest Rows in a Matrix
* Difficulty: Easy
* Tags: array, sort, search, queue, heap
*
* 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[] KWeakestRows(int[][] mat, int k) {

}
```

```
}
```

C Solution:

```
/*
* Problem: The K Weakest Rows in a Matrix
* Difficulty: Easy
* Tags: array, sort, search, queue, heap
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* kWeakestRows(int** mat, int matSize, int* matColSize, int k, int*
returnSize) {

}
```

Go Solution:

```
// Problem: The K Weakest Rows in a Matrix
// Difficulty: Easy
// Tags: array, sort, search, queue, heap
//
// 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 kWeakestRows(mat [][]int, k int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun kWeakestRows(mat: Array<IntArray>, k: Int): IntArray {
```

```
}
}
```

Swift Solution:

```
class Solution {
func kWeakestRows(_ mat: [[Int]], _ k: Int) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: The K Weakest Rows in a Matrix
// Difficulty: Easy
// Tags: array, sort, search, queue, heap
//
// 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 k_weakest_rows(mat: Vec<Vec<i32>>, k: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} mat
# @param {Integer} k
# @return {Integer[]}
def k_weakest_rows(mat, k)

end
```

PHP Solution:

```
class Solution {

/**
```

```
* @param Integer[][] $mat
* @param Integer $k
* @return Integer[]
*/
function kWeakestRows($mat, $k) {

}
}
```

Dart Solution:

```
class Solution {
List<int> kWeakestRows(List<List<int>> mat, int k) {

}
}
```

Scala Solution:

```
object Solution {
def kWeakestRows(mat: Array[Array[Int]], k: Int): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec k_weakest_rows(mat :: [[integer]], k :: integer) :: [integer]
def k_weakest_rows(mat, k) do

end
end
```

Erlang Solution:

```
-spec k_weakest_rows(Mat :: [[integer()]], K :: integer()) -> [integer()].
k_weakest_rows(Mat, K) ->
.
```

Racket Solution:

```
(define/contract (k-weakest-rows mat k)
(-> (listof (listof exact-integer?)) exact-integer? (listof exact-integer?))
)
```