Problem 2661: First Completely Painted Row or Column

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

arr

, and an

m x n

integer

matrix

mat

.

arr

and

mat

both contain

all

the integers in the range

[1, m * n]

.

Go through each index

i

in

arr

starting from index

0

and paint the cell in

mat

containing the integer

arr[i]

.

Return

the smallest index

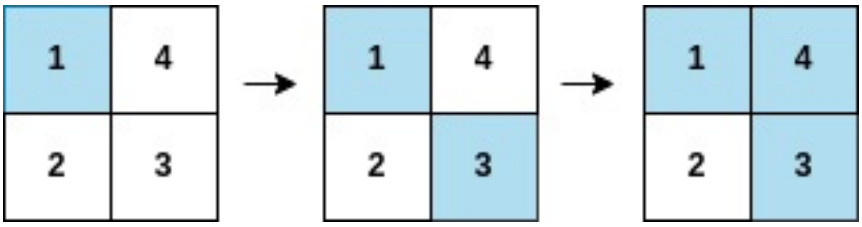
i

at which either a row or a column will be completely painted in

mat

.

Example 1:



Input:

arr = [1,3,4,2], mat = [[1,4],[2,3]]

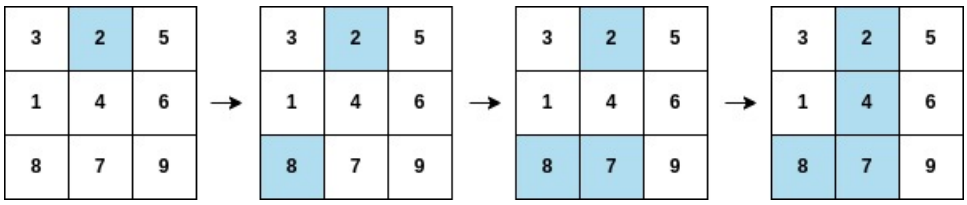
Output:

2

Explanation:

The moves are shown in order, and both the first row and second column of the matrix become fully painted at arr[2].

Example 2:



Input:

arr = [2,8,7,4,1,3,5,6,9], mat = [[3,2,5],[1,4,6],[8,7,9]]

Output:

3

Explanation:

The second column becomes fully painted at arr[3].

Constraints:

m == mat.length

n = mat[i].length

arr.length == m * n

1 <= m, n <= 10

5

1 <= m * n <= 10

5

1 <= arr[i], mat[r][c] <= m * n

All the integers of

arr

are

unique

.

All the integers of

mat

are

unique

.

Code Snippets

C++:

```
class Solution {
public:
int firstCompleteIndex(vector<int>& arr, vector<vector<int>>& mat) {

}
};
```

Java:

```
class Solution {
public int firstCompleteIndex(int[] arr, int[][] mat) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function firstCompleteIndex(arr: number[], mat: number[][]): number {

};
```

C#:

```
public class Solution {
public int FirstCompleteIndex(int[] arr, int[][] mat) {

}
}
```

C:

```
int firstCompleteIndex(int* arr, int arrSize, int** mat, int matSize, int*
matColSize) {

}
```

Go:

```
func firstCompleteIndex(arr []int, mat [][]int) int {

}
```

Kotlin:

```
class Solution {
fun firstCompleteIndex(arr: IntArray, mat: Array<IntArray>): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} arr
# @param {Integer[][]} mat
# @return {Integer}
def first_complete_index(arr, mat)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $arr
* @param Integer[][] $mat
* @return Integer
*/
function firstCompleteIndex($arr, $mat) {

}
}
```

Dart:

```
class Solution {
int firstCompleteIndex(List<int> arr, List<List<int>> mat) {
```

```
}
}
```

Scala:

```
object Solution {
def firstCompleteIndex(arr: Array[Int], mat: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec first_complete_index(arr :: [integer], mat :: [[integer]]) :: integer
def first_complete_index(arr, mat) do

end
end
```

Erlang:

```
-spec first_complete_index(Arr :: [integer()], Mat :: [[integer()]]) ->
integer().
first_complete_index(Arr, Mat) ->
.
```

Racket:

```
(define/contract (first-complete-index arr mat)
(-> (listof exact-integer?) (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: First Completely Painted Row or Column
```

```
* Difficulty: Medium
* Tags: array, hash
*
* 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 firstCompleteIndex(vector<int>& arr, vector<vector<int>>& mat) {

}
};
```

Java Solution:

```
/**
* Problem: First Completely Painted Row or Column
* Difficulty: Medium
* Tags: array, hash
*
* 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 firstCompleteIndex(int[] arr, int[][] mat) {

}
}
```

Python3 Solution:

```
"""
Problem: First Completely Painted Row or Column
Difficulty: Medium
Tags: array, hash

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: First Completely Painted Row or Column
* Difficulty: Medium
* Tags: array, hash
*
* 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[]} arr
* @param {number[][]} mat
* @return {number}
*/
var firstCompleteIndex = function(arr, mat) {

};
```

TypeScript Solution:

```
/**
* Problem: First Completely Painted Row or Column
* Difficulty: Medium
* Tags: array, hash
*
* 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 firstCompleteIndex(arr: number[], mat: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: First Completely Painted Row or Column
* Difficulty: Medium
* Tags: array, hash
*
* 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 FirstCompleteIndex(int[] arr, int[][] mat) {

}
}
```

C Solution:

```
/*
* Problem: First Completely Painted Row or Column
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

int firstCompleteIndex(int* arr, int arrSize, int** mat, int matSize, int*
matColSize) {

}
```

Go Solution:

```
// Problem: First Completely Painted Row or Column
// Difficulty: Medium
// Tags: array, hash
//
// 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 firstCompleteIndex(arr []int, mat [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun firstCompleteIndex(arr: IntArray, mat: Array<IntArray>): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: First Completely Painted Row or Column
// Difficulty: Medium
```

```
// Tags: array, hash
//
// 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 first_complete_index(arr: Vec<i32>, mat: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} arr
# @param {Integer[][]} mat
# @return {Integer}
def first_complete_index(arr, mat)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $arr
* @param Integer[][] $mat
* @return Integer
*/
function firstCompleteIndex($arr, $mat) {

}
}
```

Dart Solution:

```
class Solution {
int firstCompleteIndex(List<int> arr, List<List<int>> mat) {

}
```

```
}
```

Scala Solution:

```
object Solution {
def firstCompleteIndex(arr: Array[Int], mat: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec first_complete_index(arr :: [integer], mat :: [[integer]]) :: integer
def first_complete_index(arr, mat) do

end
end
```

Erlang Solution:

```
-spec first_complete_index(Arr :: [integer()], Mat :: [[integer()]]) ->
integer().
first_complete_index(Arr, Mat) ->
.
```

Racket Solution:

```
(define/contract (first-complete-index arr mat)
(-> (listof exact-integer?) (listof (listof exact-integer?)) exact-integer?)
)
```