Problem 1380: Lucky Numbers in a Matrix

Problem Information

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

Problem Description

Given an

m x n

matrix of

distinct

numbers, return

all

lucky numbers

in the matrix in

any

order

.

A

lucky number

is an element of the matrix such that it is the minimum element in its row and maximum in its column.

Example 1:

Input:

matrix = [[3,7,8],[9,11,13],[15,16,17]]

Output:

[15]

Explanation:

15 is the only lucky number since it is the minimum in its row and the maximum in its column.

Example 2:

Input:

matrix = [[1,10,4,2],[9,3,8,7],[15,16,17,12]]

Output:

[12]

Explanation:

12 is the only lucky number since it is the minimum in its row and the maximum in its column.

Example 3:

Input:

matrix = [[7,8],[1,2]]

Output:

[7]

Explanation:

7 is the only lucky number since it is the minimum in its row and the maximum in its column.

Constraints:

m == mat.length

n == mat[i].length

1 <= n, m <= 50

1 <= matrix[i][j] <= 10

5

.

All elements in the matrix are distinct.

Code Snippets

C++:

```
class Solution {
public:
vector<int> luckyNumbers(vector<vector<int>>& matrix) {

}
};
```

Java:

```
class Solution {
public List<Integer> luckyNumbers(int[][] matrix) {

}
```

```
}
```

Python3:

```
class Solution:
def luckyNumbers(self, matrix: List[List[int]]) -> List[int]:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function luckyNumbers(matrix: number[][]): number[] {

};
```

C#:

```
public class Solution {
public IList<int> LuckyNumbers(int[][] matrix) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* luckyNumbers(int** matrix, int matrixSize, int* matrixColSize, int*
returnSize) {

}
```

Go:

```
func luckyNumbers(matrix [][]int) []int {

}
```

Kotlin:

```
class Solution {
fun luckyNumbers(matrix: Array<IntArray>): List<Int> {

}
}
```

Swift:

```
class Solution {
func luckyNumbers(_ matrix: [[Int]]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn lucky_numbers(matrix: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[][]} matrix
# @return {Integer[]}
def lucky_numbers(matrix)
```

```
end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $matrix
* @return Integer[]
*/
function luckyNumbers($matrix) {

}
}
```

Dart:

```
class Solution {
List<int> luckyNumbers(List<List<int>> matrix) {

}
}
```

Scala:

```
object Solution {
def luckyNumbers(matrix: Array[Array[Int]]): List[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec lucky_numbers(matrix :: [[integer]]) :: [integer]
def lucky_numbers(matrix) do

end
end
```

Erlang:

```
-spec lucky_numbers(Matrix :: [[integer()]]) -> [integer()].
lucky_numbers(Matrix) ->
.
```

Racket:

```
(define/contract (lucky-numbers matrix)
(-> (listof (listof exact-integer?)) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Lucky Numbers in a Matrix
* Difficulty: Easy
* Tags: array
*
* 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> luckyNumbers(vector<vector<int>>& matrix) {

}
};
```

Java Solution:

```
/**
* Problem: Lucky Numbers in a Matrix
* Difficulty: Easy
* Tags: array
*
* 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 List<Integer> luckyNumbers(int[][] matrix) {

}
}
```

Python3 Solution:

```
"""
Problem: Lucky Numbers in a Matrix
Difficulty: Easy
Tags: array

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Lucky Numbers in a Matrix
* Difficulty: Easy
* Tags: array
*
```

```
* 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[][]} matrix
* @return {number[]}
*/
var luckyNumbers = function(matrix) {

};
```

TypeScript Solution:

```
/**
* Problem: Lucky Numbers in a Matrix
* Difficulty: Easy
* Tags: array
*
* 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 luckyNumbers(matrix: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Lucky Numbers in a Matrix
* Difficulty: Easy
* Tags: array
*
* 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 IList<int> LuckyNumbers(int[][] matrix) {

}
}
```

C Solution:

```
/*
* Problem: Lucky Numbers in a Matrix
* Difficulty: Easy
* Tags: array
*
* 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* luckyNumbers(int** matrix, int matrixSize, int* matrixColSize, int*
returnSize) {

}
```

Go Solution:

```
// Problem: Lucky Numbers in a Matrix
// Difficulty: Easy
// Tags: array
//
// 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 luckyNumbers(matrix [][]int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun luckyNumbers(matrix: Array<IntArray>): List<Int> {

}
}
```

Swift Solution:

```
class Solution {
func luckyNumbers(_ matrix: [[Int]]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Lucky Numbers in a Matrix
// Difficulty: Easy
// Tags: array
//
// 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 lucky_numbers(matrix: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} matrix
# @return {Integer[]}
def lucky_numbers(matrix)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer[][] $matrix
* @return Integer[]
*/
function luckyNumbers($matrix) {

}
}
```

Dart Solution:

```
class Solution {
List<int> luckyNumbers(List<List<int>> matrix) {

}
}
```

Scala Solution:

```
object Solution {
def luckyNumbers(matrix: Array[Array[Int]]): List[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec lucky_numbers(matrix :: [[integer]]) :: [integer]
def lucky_numbers(matrix) do

end
end
```

Erlang Solution:

```
-spec lucky_numbers(Matrix :: [[integer()]]) -> [integer()].
lucky_numbers(Matrix) ->
.
```

Racket Solution:

```
(define/contract (lucky-numbers matrix)
(-> (listof (listof exact-integer?)) (listof exact-integer?))
)
```