

Problem 2643: Row With Maximum Ones

Problem Information

Difficulty: [Easy](#)

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Given a

$m \times n$

binary matrix

`mat`

, find the

0-indexed

position of the row that contains the

maximum

count of

ones,

and the number of ones in that row.

In case there are multiple rows that have the maximum count of ones, the row with the

smallest row number

should be selected.

Return

an array containing the index of the row, and the number of ones in it.

Example 1:

Input:

mat = [[0,1],[1,0]]

Output:

[0,1]

Explanation:

Both rows have the same number of 1's. So we return the index of the smaller row, 0, and the maximum count of ones (1

)

. So, the answer is [0,1].

Example 2:

Input:

mat = [[0,0,0],[0,1,1]]

Output:

[1,2]

Explanation:

The row indexed 1 has the maximum count of ones

(2)

. So we return its index,

1

, and the count. So, the answer is [1,2].

Example 3:

Input:

```
mat = [[0,0],[1,1],[0,0]]
```

Output:

[1,2]

Explanation:

The row indexed 1 has the maximum count of ones (2). So the answer is [1,2].

Constraints:

```
m == mat.length
```

```
n == mat[i].length
```

```
1 <= m, n <= 100
```

```
mat[i][j]
```

is either

0

or

1

Code Snippets

C++:

```
class Solution {
public:
    vector<int> rowAndMaximumOnes(vector<vector<int>>& mat) {

    }
};
```

Java:

```
class Solution {
    public int[] rowAndMaximumOnes(int[][] mat) {

    }
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
 * @param {number[][]} mat
 * @return {number[]}
 */
```

```
var rowAndMaximumOnes = function(mat) {  
  
};
```

TypeScript:

```
function rowAndMaximumOnes(mat: number[][]): number[] {  
  
};
```

C#:

```
public class Solution {  
    public int[] RowAndMaximumOnes(int[][] mat) {  
  
    }  
}
```

C:

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

Go:

```
func rowAndMaximumOnes(mat [][]int) []int {  
  
}
```

Kotlin:

```
class Solution {  
    fun rowAndMaximumOnes(mat: Array<IntArray>): IntArray {  
  
    }  
}
```

Swift:

```
class Solution {  
    func rowAndMaximumOnes(_ mat: [[Int]]) -> [Int] {  
  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn row_and_maximum_ones(mat: Vec<Vec<i32>>) -> Vec<i32> {  
  
    }  
}
```

Ruby:

```
# @param {Integer[][]} mat  
# @return {Integer[]}  
def row_and_maximum_ones(mat)  
  
end
```

PHP:

```
class Solution {  
  
    /**  
     * @param Integer[][] $mat  
     * @return Integer[]  
     */  
    function rowAndMaximumOnes($mat) {  
  
    }  
}
```

Dart:

```
class Solution {  
    List<int> rowAndMaximumOnes(List<List<int>> mat) {  
  
    }  
}
```

```
}
```

Scala:

```
object Solution {  
  def rowAndMaximumOnes(mat: Array[Array[Int]]): Array[Int] = {  
  
  }  
}
```

Elixir:

```
defmodule Solution do  
  @spec row_and_maximum_ones(mat :: [[integer]]) :: [integer]  
  def row_and_maximum_ones(mat) do  
  
  end  
end
```

Erlang:

```
-spec row_and_maximum_ones(Mat :: [[integer()]]) -> [integer()].  
row_and_maximum_ones(Mat) ->  
.
```

Racket:

```
(define/contract (row-and-maximum-ones mat)  
  (-> (listof (listof exact-integer?)) (listof exact-integer?))  
)
```

Solutions

C++ Solution:

```
/*  
 * Problem: Row With Maximum Ones  
 * 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> rowAndMaximumOnes(vector<vector<int>>& mat) {

}
};

```

Java Solution:

```

/**
 * Problem: Row With Maximum Ones
 * 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 int[] rowAndMaximumOnes(int[][] mat) {

}
}

```

Python3 Solution:

```

"""
Problem: Row With Maximum Ones
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 rowAndMaximumOnes(self, mat: List[List[int]]) -> List[int]:
# TODO: Implement optimized solution
pass

```

Python Solution:

```

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

```

JavaScript Solution:

```

/**
 * Problem: Row With Maximum Ones
 * 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[][]} mat
 * @return {number[]}
 */
var rowAndMaximumOnes = function(mat) {

};

```

TypeScript Solution:

```

/**
 * Problem: Row With Maximum Ones
 * 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 rowAndMaximumOnes(mat: number[][]): number[] {

};

```

C# Solution:

```

/*
* Problem: Row With Maximum Ones
* 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 int[] RowAndMaximumOnes(int[][] mat) {

    }
}

```

C Solution:

```

/*
* Problem: Row With Maximum Ones
* 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* rowAndMaximumOnes(int** mat, int matSize, int* matColSize, int*
returnSize) {

}

```

Go Solution:

```

// Problem: Row With Maximum Ones
// 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 rowAndMaximumOnes(mat [][]int) []int {

}

```

Kotlin Solution:

```

class Solution {
    fun rowAndMaximumOnes(mat: Array<IntArray>): IntArray {

    }
}

```

Swift Solution:

```

class Solution {
    func rowAndMaximumOnes(_ mat: [[Int]]) -> [Int] {

    }
}

```

Rust Solution:

```

// Problem: Row With Maximum Ones
// 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 row_and_maximum_ones(mat: Vec<Vec<i32>>) -> Vec<i32> {

    }
}

```

Ruby Solution:

```

# @param {Integer[][]} mat
# @return {Integer[]}
def row_and_maximum_ones(mat)

end

```

PHP Solution:

```

class Solution {

    /**
     * @param Integer[][] $mat
     * @return Integer[]
     */
    function rowAndMaximumOnes($mat) {

    }

}

```

Dart Solution:

```

class Solution {
    List<int> rowAndMaximumOnes(List<List<int>> mat) {

    }

}

```

Scala Solution:

```
object Solution {  
  def rowAndMaximumOnes(mat: Array[Array[Int]]): Array[Int] = {  
  
  }  
}
```

Elixir Solution:

```
defmodule Solution do  
  @spec row_and_maximum_ones(mat :: [[integer]]) :: [integer]  
  def row_and_maximum_ones(mat) do  
  
  end  
end
```

Erlang Solution:

```
-spec row_and_maximum_ones(Mat :: [[integer()]]) -> [integer()].  
row_and_maximum_ones(Mat) ->  
.
```

Racket Solution:

```
(define/contract (row-and-maximum-ones mat)  
  (-> (listof (listof exact-integer?)) (listof exact-integer?))  
)
```