Problem 2397: Maximum Rows Covered by Columns

Problem Information

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

Problem Description

You are given an

m x n

binary matrix

matrix

and an integer

numSelect

.

Your goal is to select exactly

numSelect

distinct

columns from

matrix

such that you cover as many rows as possible.

A row is considered

covered

if all the

1

's in that row are also part of a column that you have selected. If a row does not have any

1

s, it is also considered covered.

More formally, let us consider

selected = {c

1

, c

2

, ..., c

numSelect

}

as the set of columns selected by you. A row

i

is

covered

by

selected

if:

For each cell where

matrix[i][j] == 1

, the column

j

is in

selected

.

Or, no cell in row

i

has a value of

1

.

Return the

maximum

number of rows that can be

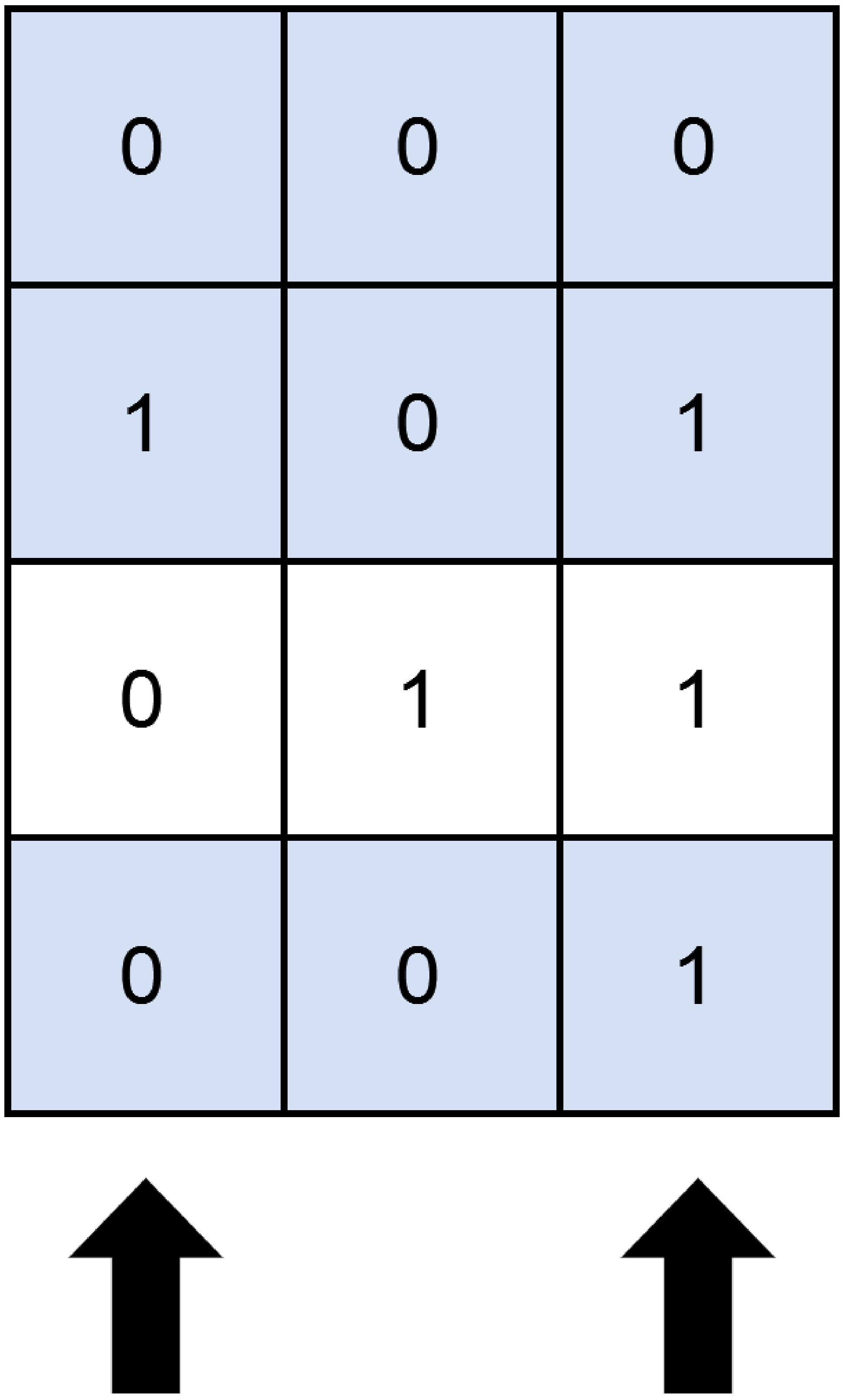
covered

by a set of

numSelect

columns.

Example 1:



Input:

matrix = [[0,0,0],[1,0,1],[0,1,1],[0,0,1]], numSelect = 2

Output:

3

Explanation:

One possible way to cover 3 rows is shown in the diagram above.

We choose s = {0, 2}.

- Row 0 is covered because it has no occurrences of 1.

- Row 1 is covered because the columns with value 1, i.e. 0 and 2 are present in s.

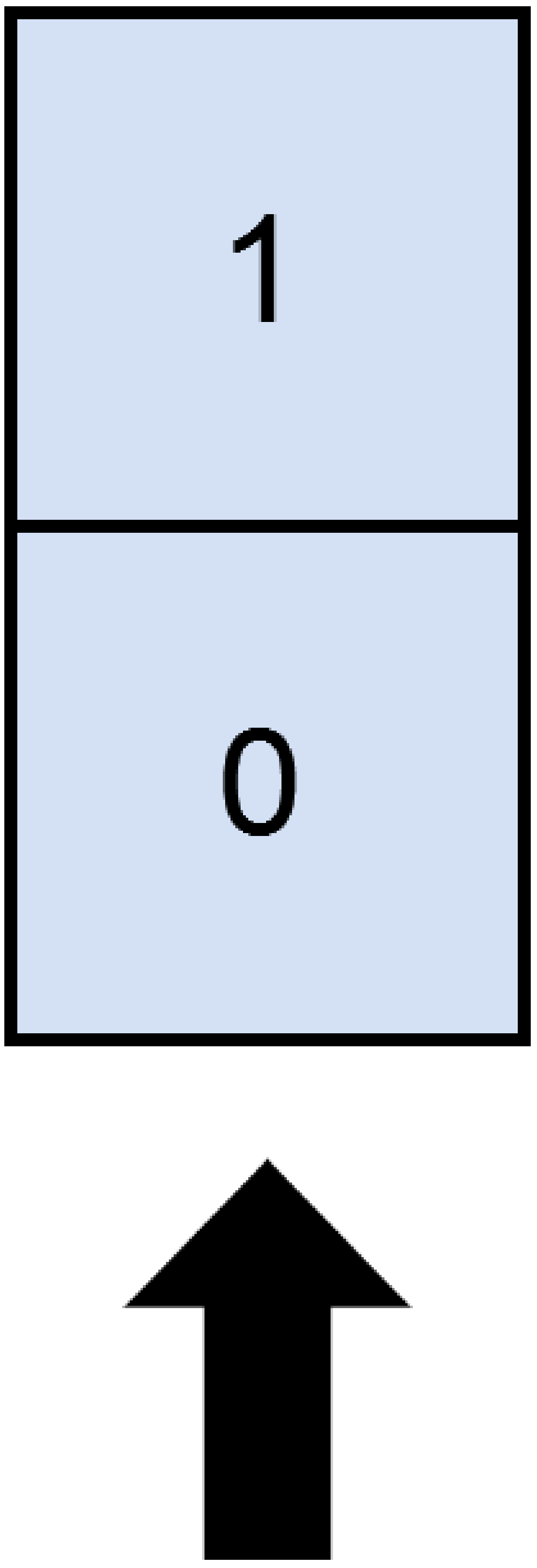
- Row 2 is not covered because matrix[2][1] == 1 but 1 is not present in s.

- Row 3 is covered because matrix[2][2] == 1 and 2 is present in s.

Thus, we can cover three rows.

Note that s = {1, 2} will also cover 3 rows, but it can be shown that no more than three rows can be covered.

Example 2:



Input:

matrix = [[1],[0]], numSelect = 1

Output:

2

Explanation:

Selecting the only column will result in both rows being covered since the entire matrix is selected.

Constraints:

m == matrix.length

n == matrix[i].length

1 <= m, n <= 12

matrix[i][j]

is either

0

or

1

.

1 <= numSelect <= n

Code Snippets

C++:

```
class Solution {
public:
int maximumRows(vector<vector<int>>& matrix, int numSelect) {
}
};
```

Java:

```
class Solution {
public int maximumRows(int[][] matrix, int numSelect) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maximumRows(matrix: number[][], numSelect: number): number {
};
```

C#:

```
public class Solution {
public int MaximumRows(int[][] matrix, int numSelect) {

}
}
```

C:

```
int maximumRows(int** matrix, int matrixSize, int* matrixColSize, int
numSelect) {

}
```

Go:

```
func maximumRows(matrix [][]int, numSelect int) int {

}
```

Kotlin:

```
class Solution {
fun maximumRows(matrix: Array<IntArray>, numSelect: Int): Int {

}
}
```

Swift:

```
class Solution {
func maximumRows(_ matrix: [[Int]], _ numSelect: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn maximum_rows(matrix: Vec<Vec<i32>>, num_select: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} matrix
# @param {Integer} num_select
# @return {Integer}
def maximum_rows(matrix, num_select)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $matrix
* @param Integer $numSelect
* @return Integer
*/
function maximumRows($matrix, $numSelect) {

}
}
```

Dart:

```
class Solution {
int maximumRows(List<List<int>> matrix, int numSelect) {

}
}
```

Scala:

```
object Solution {
def maximumRows(matrix: Array[Array[Int]], numSelect: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec maximum_rows(matrix :: [[integer]], num_select :: integer) :: integer
def maximum_rows(matrix, num_select) do

end
end
```

Erlang:

```
-spec maximum_rows(Matrix :: [[integer()]], NumSelect :: integer()) ->
integer().
maximum_rows(Matrix, NumSelect) ->
.
```

Racket:

```
(define/contract (maximum-rows matrix numSelect)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Rows Covered by Columns
* Difficulty: Medium
* 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 maximumRows(vector<vector<int>>& matrix, int numSelect) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Rows Covered by Columns
* Difficulty: Medium
* 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 maximumRows(int[][] matrix, int numSelect) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Rows Covered by Columns
Difficulty: Medium
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 maximumRows(self, matrix: List[List[int]], numSelect: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Rows Covered by Columns
* Difficulty: Medium
* 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
* @param {number} numSelect
* @return {number}
*/
var maximumRows = function(matrix, numSelect) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Rows Covered by Columns
* Difficulty: Medium
* 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 maximumRows(matrix: number[][], numSelect: number): number {
};
```

C# Solution:

```
/*
* Problem: Maximum Rows Covered by Columns
* Difficulty: Medium
* 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 MaximumRows(int[][] matrix, int numSelect) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Rows Covered by Columns
* Difficulty: Medium
* 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
*/

int maximumRows(int** matrix, int matrixSize, int* matrixColSize, int
numSelect) {

}
```

Go Solution:

```
// Problem: Maximum Rows Covered by Columns
// Difficulty: Medium
// 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 maximumRows(matrix [][]int, numSelect int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maximumRows(matrix: Array<IntArray>, numSelect: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func maximumRows(_ matrix: [[Int]], _ numSelect: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Rows Covered by Columns
// Difficulty: Medium
// 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 maximum_rows(matrix: Vec<Vec<i32>>, num_select: i32) -> i32 {

}
}
```

Ruby Solution:

```ruby
# @param {Integer[][]} matrix
# @param {Integer} num_select
# @return {Integer}
def maximum_rows(matrix, num_select)

end
```

PHP Solution:

```php
class Solution {

/**
* @param Integer[][] $matrix
* @param Integer $numSelect
* @return Integer
*/
function maximumRows($matrix, $numSelect) {

}
}
```

Dart Solution:

```dart
class Solution {
int maximumRows(List<List<int>> matrix, int numSelect) {

}
}
```

Scala Solution:

```scala
object Solution {
def maximumRows(matrix: Array[Array[Int]], numSelect: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximum_rows(matrix :: [[integer]], num_select :: integer) :: integer
def maximum_rows(matrix, num_select) do

end
end
```

Erlang Solution:

```
-spec maximum_rows(Matrix :: [[integer()]], NumSelect :: integer()) ->
integer().
maximum_rows(Matrix, NumSelect) ->
.
```

Racket Solution:

```
(define/contract (maximum-rows matrix numSelect)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```