Problem 1820: Maximum Number of Accepted Invitations

Problem Information

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

Problem Description

There are

m

boys and

n

girls in a class attending an upcoming party.

You are given an

m x n

integer matrix

grid

, where

grid[i][j]

equals

0

or

1

. If

grid[i][j] == 1

, then that means the

i

th

boy can invite the

j

th

girl to the party. A boy can invite at most

one girl

, and a girl can accept at most

one invitation

from a boy.

Return

the

maximum

possible number of accepted invitations.

Example 1:

Input:

grid = [[1,1,1], [1,0,1], [0,0,1]]

Output:

3

Explanation:

The invitations are sent as follows: - The 1

st

boy invites the 2

nd

girl. - The 2

nd

boy invites the 1

st

girl. - The 3

rd

boy invites the 3

rd

girl.

Example 2:

Input:

grid = [[1,0,1,0], [1,0,0,0], [0,0,1,0], [1,1,1,0]]

Output:

3

Explanation:

The invitations are sent as follows: -The 1

st

boy invites the 3

rd

girl. -The 2

nd

boy invites the 1

st

girl. -The 3

rd

boy invites no one. -The 4

th

boy invites the 2

nd

girl.

Constraints:

grid.length == m

grid[i].length == n

1 <= m, n <= 200

grid[i][j]

is either

0

or

1

.

Code Snippets

C++:

```
class Solution {
public:
int maximumInvitations(vector<vector<int>>& grid) {

}
};
```

Java:

```
class Solution {
public int maximumInvitations(int[][] grid) {

}
```

```
}
```

Python3:

```
class Solution:
def maximumInvitations(self, grid: List[List[int]]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maximumInvitations(grid: number[][]): number {

};
```

C#:

```
public class Solution {
public int MaximumInvitations(int[][] grid) {

}
}
```

C:

```
int maximumInvitations(int** grid, int gridSize, int* gridColSize) {

}
```

Go:

```
func maximumInvitations(grid [][]int) int {

}
```

Kotlin:

```
class Solution {
fun maximumInvitations(grid: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func maximumInvitations(_ grid: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn maximum_invitations(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} grid
# @return {Integer}
def maximum_invitations(grid)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $grid
* @return Integer
*/
function maximumInvitations($grid) {

}
}
```

Dart:

```
class Solution {
int maximumInvitations(List<List<int>> grid) {

}
}
```

Scala:

```
object Solution {
def maximumInvitations(grid: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec maximum_invitations(grid :: [[integer]]) :: integer
def maximum_invitations(grid) do

end
end
```

Erlang:

```
-spec maximum_invitations(Grid :: [[integer()]]) -> integer().
maximum_invitations(Grid) ->
.
```

Racket:

```
(define/contract (maximum-invitations grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Number of Accepted Invitations
* Difficulty: Medium
* Tags: array, graph, search
*
* 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 maximumInvitations(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Number of Accepted Invitations
* Difficulty: Medium
* Tags: array, graph, search
*
* 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 maximumInvitations(int[][] grid) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Maximum Number of Accepted Invitations
Difficulty: Medium
Tags: array, graph, search

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Number of Accepted Invitations
* Difficulty: Medium
* Tags: array, graph, search
*
* 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[][]} grid
* @return {number}
*/
var maximumInvitations = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Number of Accepted Invitations
* Difficulty: Medium
* Tags: array, graph, search
*
* 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 maximumInvitations(grid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Number of Accepted Invitations
* Difficulty: Medium
* Tags: array, graph, search
*
* 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 MaximumInvitations(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Number of Accepted Invitations
* Difficulty: Medium
* Tags: array, graph, search
*
* 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 maximumInvitations(int** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```
// Problem: Maximum Number of Accepted Invitations
// Difficulty: Medium
// Tags: array, graph, search
//
// 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 maximumInvitations(grid [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maximumInvitations(grid: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func maximumInvitations(_ grid: [[Int]]) -> Int {
```

```
}
}
```

Rust Solution:

```
// Problem: Maximum Number of Accepted Invitations
// Difficulty: Medium
// Tags: array, graph, search
//
// 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_invitations(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} grid
# @return {Integer}
def maximum_invitations(grid)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $grid
* @return Integer
*/
function maximumInvitations($grid) {

}
}
```

Dart Solution:

```
class Solution {
int maximumInvitations(List<List<int>> grid) {

}
}
```

Scala Solution:

```
object Solution {
def maximumInvitations(grid: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximum_invitations(grid :: [[integer]]) :: integer
def maximum_invitations(grid) do

end
end
```

Erlang Solution:

```
-spec maximum_invitations(Grid :: [[integer()]]) -> integer().
maximum_invitations(Grid) ->
.
```

Racket Solution:

```
(define/contract (maximum-invitations grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```