Problem 2132: Stamping the Grid

Problem Information

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

Problem Description

You are given an

m x n

binary matrix

grid

where each cell is either

0

(empty) or

1

(occupied).

You are then given stamps of size

stampHeight x stampWidth

. We want to fit the stamps such that they follow the given

restrictions

and

requirements

:

Cover all the

empty

cells.

Do not cover any of the

occupied

cells.

We can put as

many

stamps as we want.

Stamps can

overlap

with each other.

Stamps are not allowed to be

rotated

.

Stamps must stay completely

inside

the grid.

Return

true

if it is possible to fit the stamps while following the given restrictions and requirements. Otherwise, return

false

.

Example 1:

Input:

grid = [[1,0,0,0],[1,0,0,0],[1,0,0,0],[1,0,0,0],[1,0,0,0]], stampHeight = 4, stampWidth = 3

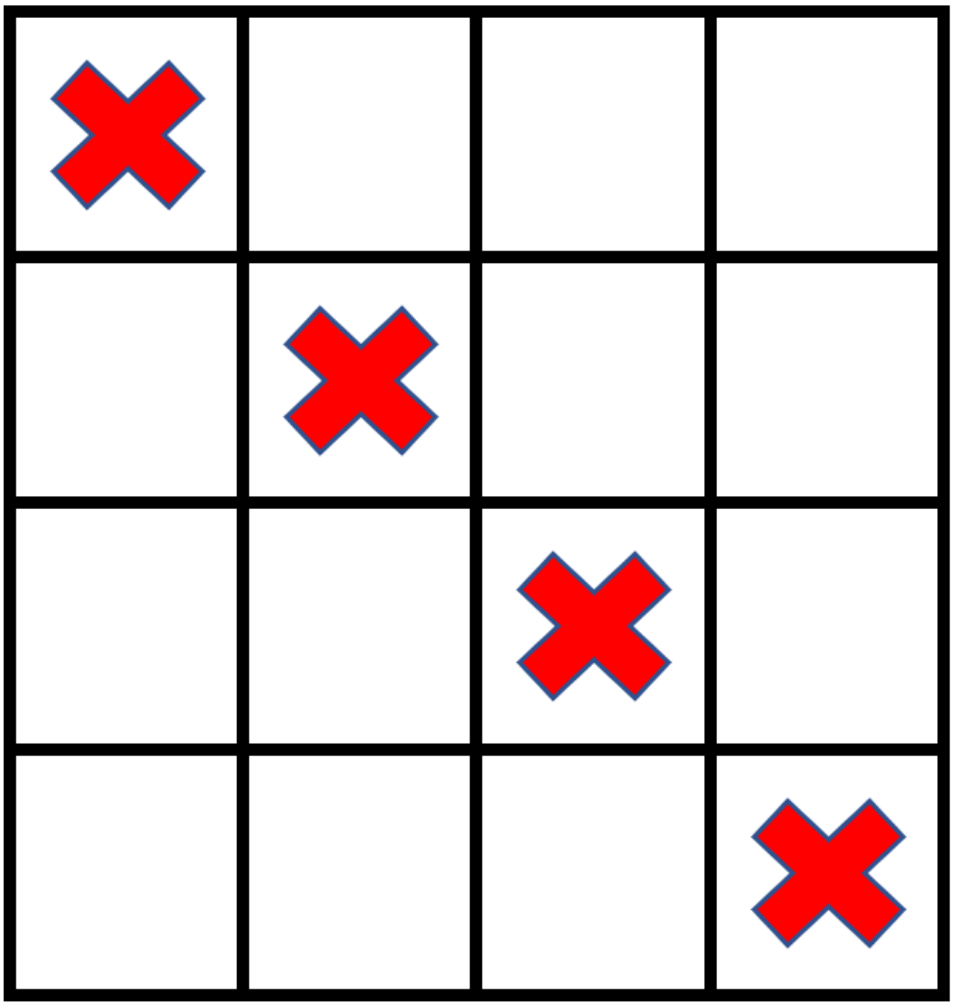
Output:

true

Explanation:

We have two overlapping stamps (labeled 1 and 2 in the image) that are able to cover all the empty cells.

Example 2:



Input:

grid = [[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]], stampHeight = 2, stampWidth = 2

Output:

false

Explanation:

There is no way to fit the stamps onto all the empty cells without the stamps going outside the grid.

Constraints:

m == grid.length

n == grid[r].length

1 <= m, n <= 10

5

1 <= m * n <= 2 * 10

5

grid[r][c]

is either

0

or

1

.

1 <= stampHeight, stampWidth <= 10

5

Code Snippets

C++:

```
class Solution {
public:
bool possibleToStamp(vector<vector<int>>& grid, int stampHeight, int
stampWidth) {
```

```
}
};
```

Java:

```
class Solution {
public boolean possibleToStamp(int[][] grid, int stampHeight, int stampWidth)
{

}
}
```

Python3:

```
class Solution:
def possibleToStamp(self, grid: List[List[int]], stampHeight: int,
stampWidth: int) -> bool:
```

Python:

```
class Solution(object):
def possibleToStamp(self, grid, stampHeight, stampWidth):
"""
:type grid: List[List[int]]
:type stampHeight: int
:type stampWidth: int
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {number[][]} grid
* @param {number} stampHeight
* @param {number} stampWidth
* @return {boolean}
*/
var possibleToStamp = function(grid, stampHeight, stampWidth) {

};
```

TypeScript:

```
function possibleToStamp(grid: number[][], stampHeight: number, stampWidth:
number): boolean {

};
```

C#:

```
public class Solution {
public bool PossibleToStamp(int[][] grid, int stampHeight, int stampWidth) {

}
}
```

C:

```
bool possibleToStamp(int** grid, int gridSize, int* gridColSize, int
stampHeight, int stampWidth) {

}
```

Go:

```
func possibleToStamp(grid [][]int, stampHeight int, stampWidth int) bool {

}
```

Kotlin:

```
class Solution {
fun possibleToStamp(grid: Array<IntArray>, stampHeight: Int, stampWidth:
Int): Boolean {

}
}
```

Swift:

```
class Solution {
func possibleToStamp(_ grid: [[Int]], _ stampHeight: Int, _ stampWidth: Int)
-> Bool {
```

```
}
}
```

Rust:

```
impl Solution {
pub fn possible_to_stamp(grid: Vec<Vec<i32>>, stamp_height: i32, stamp_width:
i32) -> bool {

}
}
```

Ruby:

```
# @param {Integer[][]} grid
# @param {Integer} stamp_height
# @param {Integer} stamp_width
# @return {Boolean}
def possible_to_stamp(grid, stamp_height, stamp_width)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $grid
* @param Integer $stampHeight
* @param Integer $stampWidth
* @return Boolean
*/
function possibleToStamp($grid, $stampHeight, $stampWidth) {

}
}
```

Dart:

```
class Solution {
bool possibleToStamp(List<List<int>> grid, int stampHeight, int stampWidth) {
```

```
}
}
```

Scala:

```
object Solution {
def possibleToStamp(grid: Array[Array[Int]], stampHeight: Int, stampWidth:
Int): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec possible_to_stamp(grid :: [[integer]], stamp_height :: integer,
stamp_width :: integer) :: boolean
def possible_to_stamp(grid, stamp_height, stamp_width) do

end
end
```

Erlang:

```
-spec possible_to_stamp(Grid :: [[integer()]], StampHeight :: integer(),
StampWidth :: integer()) -> boolean().
possible_to_stamp(Grid, StampHeight, StampWidth) ->
.
```

Racket:

```
(define/contract (possible-to-stamp grid stampHeight stampWidth)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Stamping the Grid
* Difficulty: Hard
* Tags: array, greedy
*
* 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:
bool possibleToStamp(vector<vector<int>>& grid, int stampHeight, int
stampWidth) {

}
};
```

Java Solution:

```
/**
* Problem: Stamping the Grid
* Difficulty: Hard
* Tags: array, greedy
*
* 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 boolean possibleToStamp(int[][] grid, int stampHeight, int stampWidth)
{

}
}
```

Python3 Solution:

```
"""
Problem: Stamping the Grid
Difficulty: Hard
```

```
Tags: array, greedy

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 possibleToStamp(self, grid: List[List[int]], stampHeight: int,
stampWidth: int) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def possibleToStamp(self, grid, stampHeight, stampWidth):
"""
:type grid: List[List[int]]
:type stampHeight: int
:type stampWidth: int
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Stamping the Grid
* Difficulty: Hard
* Tags: array, greedy
*
* 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
* @param {number} stampHeight
* @param {number} stampWidth
* @return {boolean}
```

```
*/
var possibleToStamp = function(grid, stampHeight, stampWidth) {

};
```

TypeScript Solution:

```
/**
* Problem: Stamping the Grid
* Difficulty: Hard
* Tags: array, greedy
*
* 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 possibleToStamp(grid: number[][], stampHeight: number, stampWidth:
number): boolean {

};
```

C# Solution:

```
/*
* Problem: Stamping the Grid
* Difficulty: Hard
* Tags: array, greedy
*
* 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 bool PossibleToStamp(int[][] grid, int stampHeight, int stampWidth) {

}
}
```

C Solution:

```
/*
* Problem: Stamping the Grid
* Difficulty: Hard
* Tags: array, greedy
*
* 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
*/

bool possibleToStamp(int** grid, int gridSize, int* gridColSize, int
stampHeight, int stampWidth) {

}
```

Go Solution:

```
// Problem: Stamping the Grid
// Difficulty: Hard
// Tags: array, greedy
//
// 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 possibleToStamp(grid [][]int, stampHeight int, stampWidth int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun possibleToStamp(grid: Array<IntArray>, stampHeight: Int, stampWidth:
Int): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func possibleToStamp(_ grid: [[Int]], _ stampHeight: Int, _ stampWidth: Int)
```

```
-> Bool {

}
}
```

Rust Solution:

```
// Problem: Stamping the Grid
// Difficulty: Hard
// Tags: array, greedy
//
// 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 possible_to_stamp(grid: Vec<Vec<i32>>, stamp_height: i32, stamp_width:
i32) -> bool {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} grid
# @param {Integer} stamp_height
# @param {Integer} stamp_width
# @return {Boolean}
def possible_to_stamp(grid, stamp_height, stamp_width)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $grid
* @param Integer $stampHeight
* @param Integer $stampWidth
* @return Boolean
*/
```

```
function possibleToStamp($grid, $stampHeight, $stampWidth) {

}
}
```

Dart Solution:

```
class Solution {
bool possibleToStamp(List<List<int>> grid, int stampHeight, int stampWidth) {

}
}
```

Scala Solution:

```
object Solution {
def possibleToStamp(grid: Array[Array[Int]], stampHeight: Int, stampWidth:
Int): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec possible_to_stamp(grid :: [[integer]], stamp_height :: integer,
stamp_width :: integer) :: boolean
def possible_to_stamp(grid, stamp_height, stamp_width) do

end
end
```

Erlang Solution:

```
-spec possible_to_stamp(Grid :: [[integer()]], StampHeight :: integer(),
StampWidth :: integer()) -> boolean().
possible_to_stamp(Grid, StampHeight, StampWidth) ->
.
```

Racket Solution:

```
(define/contract (possible-to-stamp grid stampHeight stampWidth)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer? boolean?)
)
```