Problem 221: Maximal Square

Problem Information

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

Problem Description

Given an

m x n

binary

matrix

filled with

0

's and

1

's,

find the largest square containing only

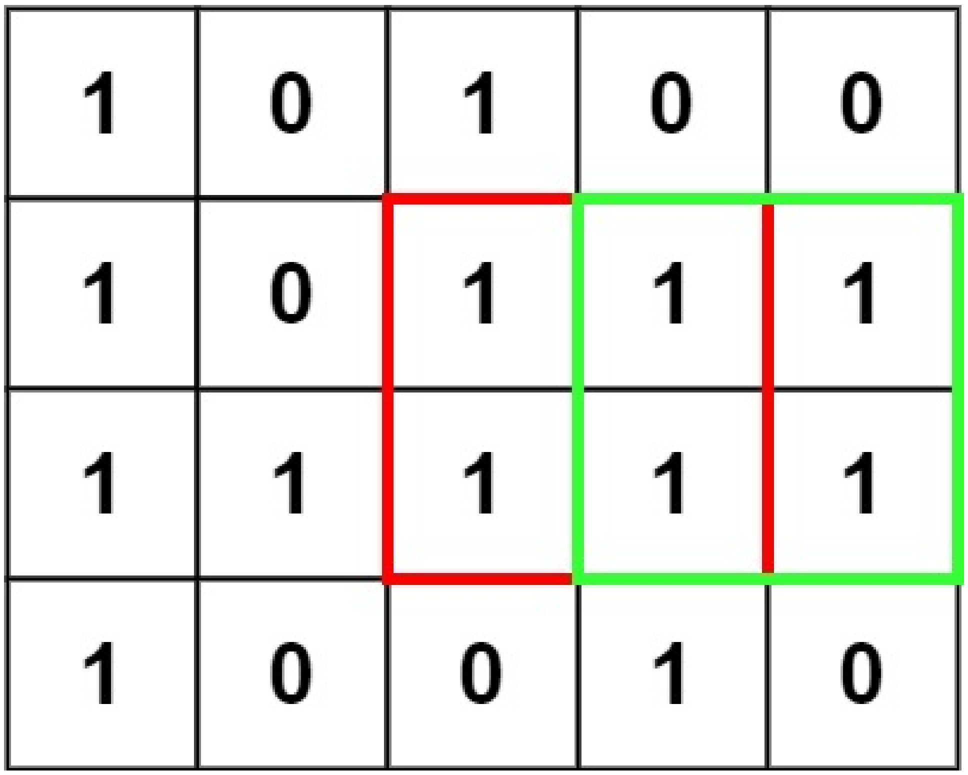
1

's

and return its area

.

Example 1:



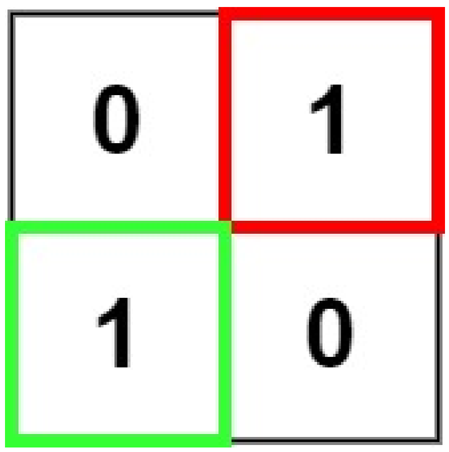
Input:

matrix = [["1","0","1","0","0"],["1","0","1","1","1"],["1","1","1","1","1"],["1","0","0","1","0"]]

Output:

4

Example 2:



Input:

matrix = [["0","1"],["1","0"]]

Output:

1

Example 3:

Input:

matrix = [["0"]]

Output:

0

Constraints:

m == matrix.length

n == matrix[i].length

1 <= m, n <= 300

matrix[i][j]

is

'0'

or

'1'

.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int maximalSquare(char[][] matrix) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maximalSquare(matrix: string[][]): number {

};
```

C#:

```
public class Solution {
public int MaximalSquare(char[][] matrix) {

}
}
```

C:

```
int maximalSquare(char** matrix, int matrixSize, int* matrixColSize) {

}
```

Go:

```
func maximalSquare(matrix [][]byte) int {

}
```

Kotlin:

```
class Solution {
fun maximalSquare(matrix: Array<CharArray>): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
int maximalSquare(List<List<String>> matrix) {

}
}
```

Scala:

```
object Solution {
def maximalSquare(matrix: Array[Array[Char]]): Int = {

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (maximal-square matrix)
(-> (listof (listof char?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximal Square
* Difficulty: Medium
* Tags: array, dp
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int maximalSquare(vector<vector<char>>& matrix) {

}
};
```

Java Solution:

```
/**
* Problem: Maximal Square
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int maximalSquare(char[][] matrix) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximal Square
Difficulty: Medium
Tags: array, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""
```

```
class Solution:
def maximalSquare(self, matrix: List[List[str]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximal Square
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Maximal Square
* Difficulty: Medium
* Tags: array, dp
```

```
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function maximalSquare(matrix: string[][]): number {

};
```

C# Solution:

```
/*
* Problem: Maximal Square
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int MaximalSquare(char[][] matrix) {

}
}
```

C Solution:

```
/*
* Problem: Maximal Square
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int maximalSquare(char** matrix, int matrixSize, int* matrixColSize) {
}
```

Go Solution:

```
// Problem: Maximal Square
// Difficulty: Medium
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func maximalSquare(matrix [][]byte) int {

}
```

Kotlin Solution:

```
class Solution {
fun maximalSquare(matrix: Array<CharArray>): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximal Square
// Difficulty: Medium
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn maximal_square(matrix: Vec<Vec<char>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
int maximalSquare(List<List<String>> matrix) {

}
}
```

Scala Solution:

```
object Solution {
def maximalSquare(matrix: Array[Array[Char]]): Int = {
}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (maximal-square matrix)
(-> (listof (listof char?)) exact-integer?)
)
```