Problem 240: Search a 2D Matrix II

Problem Information

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

Problem Description

Write an efficient algorithm that searches for a value

target

in an

m x n

integer matrix

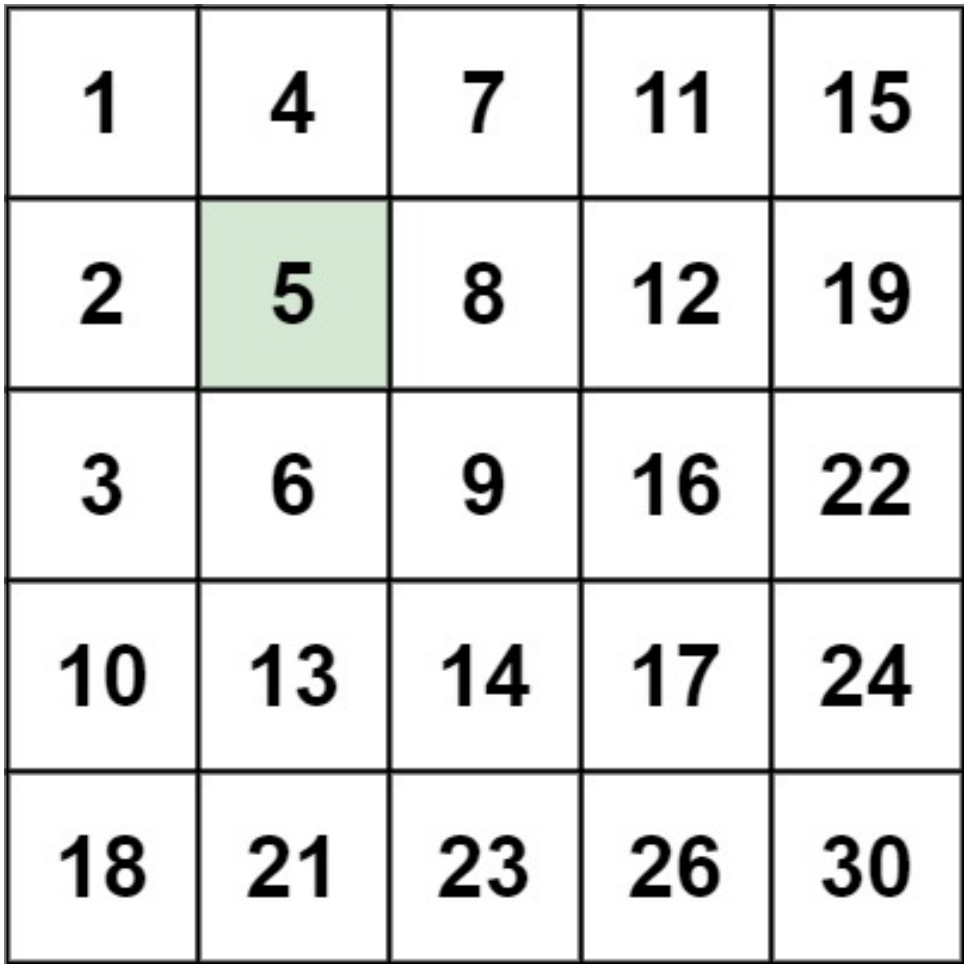
matrix

. This matrix has the following properties:

Integers in each row are sorted in ascending from left to right.

Integers in each column are sorted in ascending from top to bottom.

Example 1:



Input:

matrix = [[1,4,7,11,15],[2,5,8,12,19],[3,6,9,16,22],[10,13,14,17,24],[18,21,23,26,30]], target = 5

Output:

true

Example 2:

1	4	7	11	15
2	5	8	12	19
3	6	9	16	22
10	13	14	17	24
18	21	23	26	30

Input:

matrix = [[1,4,7,11,15],[2,5,8,12,19],[3,6,9,16,22],[10,13,14,17,24],[18,21,23,26,30]], target = 20

Output:

false

Constraints:

m == matrix.length

n == matrix[i].length

1 <= n, m <= 300

-10

9

<= matrix[i][j] <= 10

9

All the integers in each row are

sorted

in ascending order.

All the integers in each column are

sorted

in ascending order.

-10

9

<= target <= 10

9

Code Snippets

C++:

```
class Solution {
public:
bool searchMatrix(vector<vector<int>>& matrix, int target) {

}
};
```

Java:

```
class Solution {
public boolean searchMatrix(int[][] matrix, int target) {

}
}
```

Python3:

```
class Solution:
def searchMatrix(self, matrix: List[List[int]], target: int) -> bool:
```

Python:

```
class Solution(object):
def searchMatrix(self, matrix, target):
"""
:type matrix: List[List[int]]
:type target: int
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {number[][]} matrix
* @param {number} target
* @return {boolean}
*/
var searchMatrix = function(matrix, target) {

};
```

TypeScript:

```
function searchMatrix(matrix: number[][], target: number): boolean {

};
```

C#:

```
public class Solution {
public bool SearchMatrix(int[][] matrix, int target) {
```

```
}
}
```

C:

```
bool searchMatrix(int** matrix, int matrixSize, int* matrixColSize, int
target){

}
```

Go:

```
func searchMatrix(matrix [][]int, target int) bool {

}
```

Kotlin:

```
class Solution {
fun searchMatrix(matrix: Array<IntArray>, target: Int): Boolean {

}
}
```

Swift:

```
class Solution {
func searchMatrix(_ matrix: [[Int]], _ target: Int) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn search_matrix(matrix: Vec<Vec<i32>>, target: i32) -> bool {

}
}
```

Ruby:

```ruby
# @param {Integer[][]} matrix
# @param {Integer} target
# @return {Boolean}
def search_matrix(matrix, target)

end
```

PHP:

```php
class Solution {

/**
* @param Integer[][] $matrix
* @param Integer $target
* @return Boolean
*/
function searchMatrix($matrix, $target) {

}
}
```

Scala:

```scala
object Solution {
def searchMatrix(matrix: Array[Array[Int]], target: Int): Boolean = {

}
}
```

Solutions

C++ Solution:

```cpp
/*
* Problem: Search a 2D Matrix II
* Difficulty: Medium
* Tags: array, sort, 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:
bool searchMatrix(vector<vector<int>>& matrix, int target) {

}
};
```

Java Solution:

```
/**
* Problem: Search a 2D Matrix II
* Difficulty: Medium
* Tags: array, sort, 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 boolean searchMatrix(int[][] matrix, int target) {

}
}
```

Python3 Solution:

```
"""
Problem: Search a 2D Matrix II
Difficulty: Medium
Tags: array, sort, 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 searchMatrix(self, matrix: List[List[int]], target: int) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def searchMatrix(self, matrix, target):
"""
:type matrix: List[List[int]]
:type target: int
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Search a 2D Matrix II
* Difficulty: Medium
* Tags: array, sort, 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[][]} matrix
* @param {number} target
* @return {boolean}
*/
var searchMatrix = function(matrix, target) {

};
```

TypeScript Solution:

```
/**
* Problem: Search a 2D Matrix II
* Difficulty: Medium
```

```
* Tags: array, sort, 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 searchMatrix(matrix: number[][], target: number): boolean {

};
```

C# Solution:

```
/*
* Problem: Search a 2D Matrix II
* Difficulty: Medium
* Tags: array, sort, 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 bool SearchMatrix(int[][] matrix, int target) {

}
}
```

C Solution:

```
/*
* Problem: Search a 2D Matrix II
* Difficulty: Medium
* Tags: array, sort, 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
*/
```

```
bool searchMatrix(int** matrix, int matrixSize, int* matrixColSize, int
target){

}
```

Go Solution:

```
// Problem: Search a 2D Matrix II
// Difficulty: Medium
// Tags: array, sort, 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 searchMatrix(matrix [][]int, target int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun searchMatrix(matrix: Array<IntArray>, target: Int): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func searchMatrix(_ matrix: [[Int]], _ target: Int) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Search a 2D Matrix II
// Difficulty: Medium
```

```
// Tags: array, sort, 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 search_matrix(matrix: Vec<Vec<i32>>, target: i32) -> bool {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} matrix
# @param {Integer} target
# @return {Boolean}
def search_matrix(matrix, target)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $matrix
* @param Integer $target
* @return Boolean
*/
function searchMatrix($matrix, $target) {

}
}
```

Scala Solution:

```
object Solution {
def searchMatrix(matrix: Array[Array[Int]], target: Int): Boolean = {

}
```

```
}
```