Problem 1886: Determine Whether Matrix Can Be Obtained By Rotation

Problem Information

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

Problem Description

Given two

n x n

binary matrices

mat

and

target

, return

true

if it is possible to make

mat

equal to

target

by

rotating

mat

in

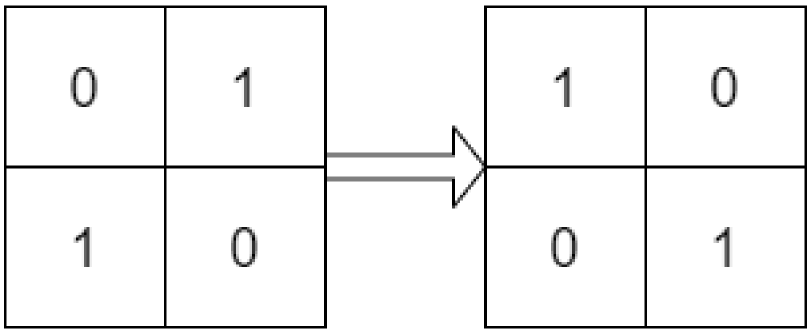
90-degree increments

, or

false

otherwise.

Example 1:



Input:

mat = [[0,1],[1,0]], target = [[1,0],[0,1]]

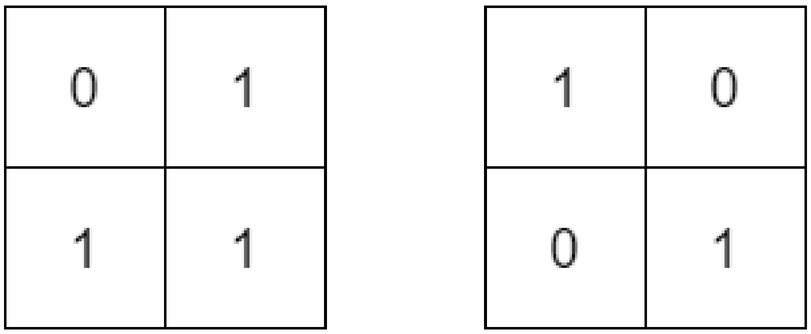
Output:

true

Explanation:

We can rotate mat 90 degrees clockwise to make mat equal target.

Example 2:



Input:

mat = [[0,1],[1,1]], target = [[1,0],[0,1]]

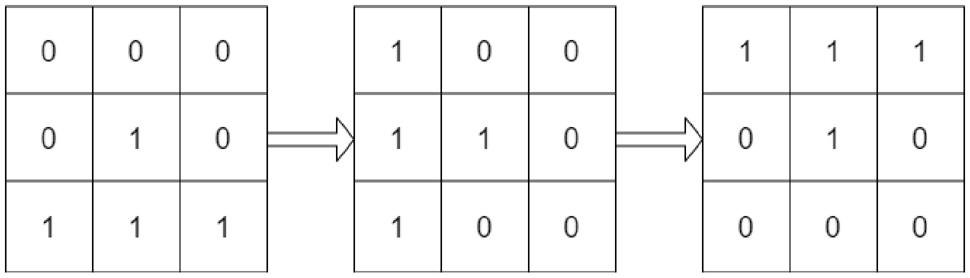
Output:

false

Explanation:

It is impossible to make mat equal to target by rotating mat.

Example 3:



Input:

mat = [[0,0,0],[0,1,0],[1,1,1]], target = [[1,1,1],[0,1,0],[0,0,0]]

Output:

true

Explanation:

We can rotate mat 90 degrees clockwise two times to make mat equal target.

Constraints:

n == mat.length == target.length

n == mat[i].length == target[i].length

1 <= n <= 10

mat[i][j]

and

target[i][j]

are either

0

or

1

.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public boolean findRotation(int[][] mat, int[][] target) {
}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function findRotation(mat: number[][], target: number[][]): boolean {

};
```

C#:

```
public class Solution {
public bool FindRotation(int[][] mat, int[][] target) {
}
}
```

C:

```
bool findRotation(int** mat, int matSize, int* matColSize, int** target, int
targetSize, int* targetColSize) {

}
```

Go:

```
func findRotation(mat [][]int, target [][]int) bool {

}
```

Kotlin:

```
class Solution {
fun findRotation(mat: Array<IntArray>, target: Array<IntArray>): Boolean {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
bool findRotation(List<List<int>> mat, List<List<int>> target) {

}
}
```

Scala:

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

}
}
```

Elixir:

```
defmodule Solution do
@spec find_rotation(mat :: [[integer]], target :: [[integer]]) :: boolean
def find_rotation(mat, target) do
```

```
end
end
```

Erlang:

```
-spec find_rotation(Mat :: [[integer()]], Target :: [[integer()]]) ->
boolean().
find_rotation(Mat, Target) ->
.
```

Racket:

```
(define/contract (find-rotation mat target)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?))
boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Determine Whether Matrix Can Be Obtained By Rotation
* Difficulty: Easy
* 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:
bool findRotation(vector<vector<int>>& mat, vector<vector<int>>& target) {

}
};
```

Java Solution:

```
/**
* Problem: Determine Whether Matrix Can Be Obtained By Rotation
* Difficulty: Easy
* 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 boolean findRotation(int[][] mat, int[][] target) {

}
}
```

Python3 Solution:

```
"""
Problem: Determine Whether Matrix Can Be Obtained By Rotation
Difficulty: Easy
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 findRotation(self, mat: List[List[int]], target: List[List[int]]) ->
bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Determine Whether Matrix Can Be Obtained By Rotation
* Difficulty: Easy
* 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[][]} mat
* @param {number[][]} target
* @return {boolean}
*/
var findRotation = function(mat, target) {

};
```

TypeScript Solution:

```
/**
* Problem: Determine Whether Matrix Can Be Obtained By Rotation
* Difficulty: Easy
* 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 findRotation(mat: number[][], target: number[][]): boolean {

};
```

C# Solution:

```
/*
* Problem: Determine Whether Matrix Can Be Obtained By Rotation
* Difficulty: Easy
* 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 bool FindRotation(int[][] mat, int[][] target) {

}
}
```

C Solution:

```
/*
* Problem: Determine Whether Matrix Can Be Obtained By Rotation
* Difficulty: Easy
* 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
*/

bool findRotation(int** mat, int matSize, int* matColSize, int** target, int
targetSize, int* targetColSize) {

}
```

Go Solution:

```
// Problem: Determine Whether Matrix Can Be Obtained By Rotation
// Difficulty: Easy
// 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 findRotation(mat [][]int, target [][]int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun findRotation(mat: Array<IntArray>, target: Array<IntArray>): Boolean {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Determine Whether Matrix Can Be Obtained By Rotation
// Difficulty: Easy
// 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 find_rotation(mat: Vec<Vec<i32>>, target: Vec<Vec<i32>>) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
bool findRotation(List<List<int>> mat, List<List<int>> target) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_rotation(mat :: [[integer]], target :: [[integer]]) :: boolean
def find_rotation(mat, target) do
```

```
end
end
```

Erlang Solution:

```
-spec find_rotation(Mat :: [[integer()]], Target :: [[integer()]]) ->
boolean().
find_rotation(Mat, Target) ->
.
```

Racket Solution:

```
(define/contract (find-rotation mat target)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?))
boolean?)
)
```