Problem 2946: Matrix Similarity After Cyclic Shifts

Problem Information

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

Problem Description

You are given an

m x n

integer matrix

mat

and an integer

k

. The matrix rows are 0-indexed.

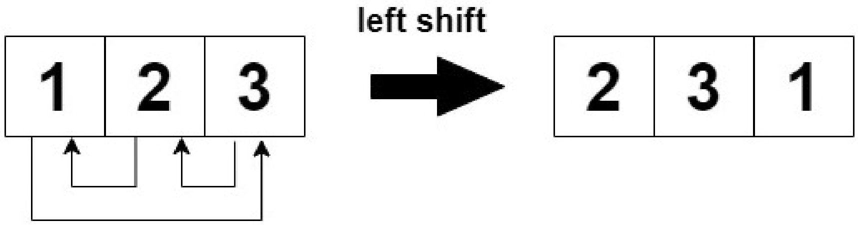
The following proccess happens

k

times:

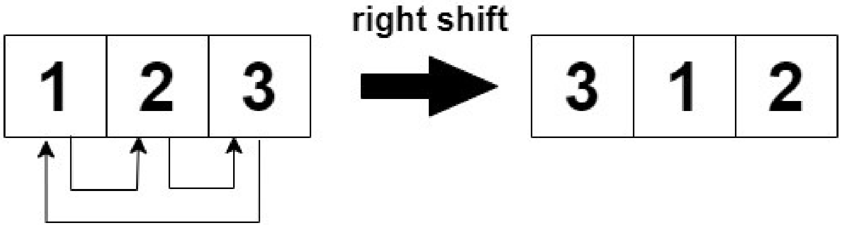
Even-indexed

rows (0, 2, 4, ...) are cyclically shifted to the left.



Odd-indexed

rows (1, 3, 5, ...) are cyclically shifted to the right.



Return

true

if the final modified matrix after

k

steps is identical to the original matrix, and

false

otherwise.

Example 1:

Input:

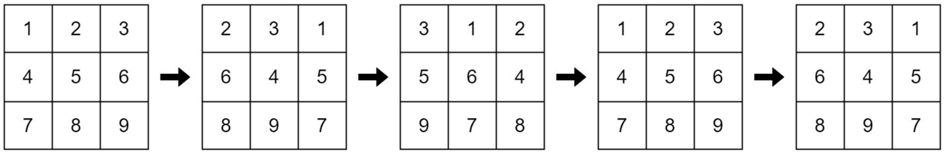
mat = [[1,2,3],[4,5,6],[7,8,9]], k = 4

Output:

false

Explanation:

In each step left shift is applied to rows 0 and 2 (even indices), and right shift to row 1 (odd index).



Example 2:

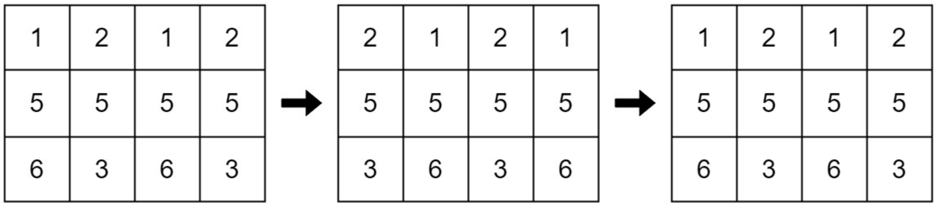
Input:

mat = [[1,2,1,2],[5,5,5,5],[6,3,6,3]], k = 2

Output:

true

Explanation:



Example 3:

Input:

mat = [[2,2],[2,2]], k = 3

Output:

true

Explanation:

As all the values are equal in the matrix, even after performing cyclic shifts the matrix will remain the same.

Constraints:

1 <= mat.length <= 25

1 <= mat[i].length <= 25

1 <= mat[i][j] <= 25

1 <= k <= 50

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public boolean areSimilar(int[][] mat, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function areSimilar(mat: number[][], k: number): boolean {

};
```

C#:

```
public class Solution {
public bool AreSimilar(int[][] mat, int k) {

}
}
```

C:

```
bool areSimilar(int** mat, int matSize, int* matColSize, int k) {

}
```

Go:

```
func areSimilar(mat [][]int, k int) bool {

}
```

Kotlin:

```
class Solution {
fun areSimilar(mat: Array<IntArray>, k: Int): Boolean {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
bool areSimilar(List<List<int>> mat, int k) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec are_similar(Mat :: [[integer()]], K :: integer()) -> boolean().
are_similar(Mat, K) ->
.
```

Racket:

```
(define/contract (are-similar mat k)
(-> (listof (listof exact-integer?)) exact-integer? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Matrix Similarity After Cyclic Shifts
* Difficulty: Easy
* Tags: array, math
*
* 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 areSimilar(vector<vector<int>>& mat, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Matrix Similarity After Cyclic Shifts
* Difficulty: Easy
* Tags: array, math
*
* 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 areSimilar(int[][] mat, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Matrix Similarity After Cyclic Shifts
Difficulty: Easy
Tags: array, math

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 areSimilar(self, mat: List[List[int]], k: int) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Matrix Similarity After Cyclic Shifts
* Difficulty: Easy
* Tags: array, math
*
* 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} k
* @return {boolean}
*/
var areSimilar = function(mat, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Matrix Similarity After Cyclic Shifts
* Difficulty: Easy
* Tags: array, math
*
* 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 areSimilar(mat: number[][], k: number): boolean {

};
```

C# Solution:

```
/*
* Problem: Matrix Similarity After Cyclic Shifts
* Difficulty: Easy
* Tags: array, math
*
* 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 AreSimilar(int[][] mat, int k) {

}

}
```

C Solution:

```
/*
* Problem: Matrix Similarity After Cyclic Shifts
* Difficulty: Easy
* Tags: array, math
*
* 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 areSimilar(int** mat, int matSize, int* matColSize, int k) {

}
```

Go Solution:

```
// Problem: Matrix Similarity After Cyclic Shifts
// Difficulty: Easy
// Tags: array, math
//
// 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 areSimilar(mat [][]int, k int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun areSimilar(mat: Array<IntArray>, k: Int): Boolean {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Matrix Similarity After Cyclic Shifts
// Difficulty: Easy
// Tags: array, math
//
// 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 are_similar(mat: Vec<Vec<i32>>, k: i32) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
bool areSimilar(List<List<int>> mat, int k) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec are_similar(Mat :: [[integer()]], K :: integer()) -> boolean().
are_similar(Mat, K) ->
.
```

Racket Solution:

```
(define/contract (are-similar mat k)
(-> (listof (listof exact-integer?)) exact-integer? boolean?)
)
```