Problem 1914: Cyclically Rotating a Grid

Problem Information

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

Problem Description

You are given an

m x n

integer matrix

grid

, where

m

and

n

are both

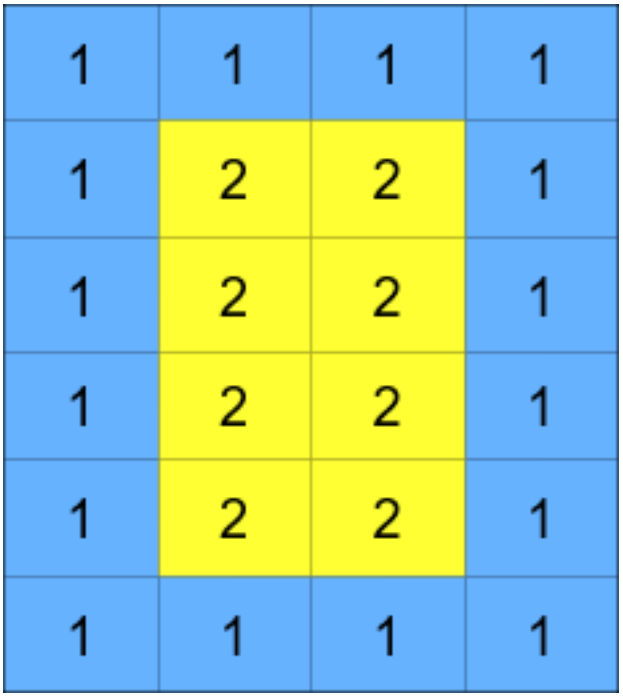
even

integers, and an integer

k

.

The matrix is composed of several layers, which is shown in the below image, where each color is its own layer:



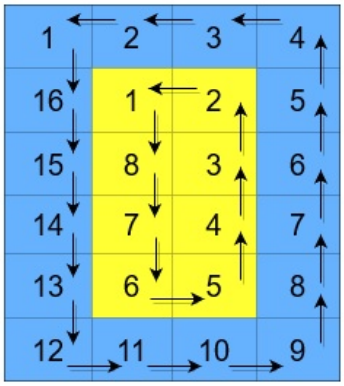
A cyclic rotation of the matrix is done by cyclically rotating

each layer

in the matrix. To cyclically rotate a layer once, each element in the layer will take the place of the adjacent element in the

counter-clockwise

direction. An example rotation is shown below:



Before rotation

After rotation with k = 1

Return

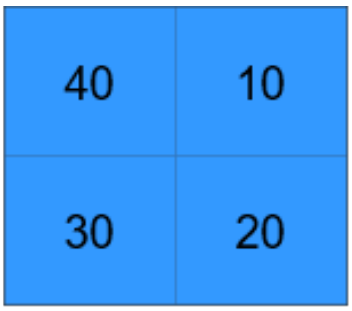
the matrix after applying

k

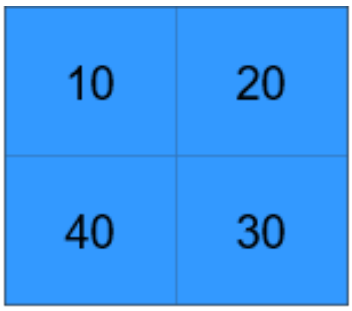
cyclic rotations to it

.

Example 1:



Before Any Rotations



After One Rotation

Input:

grid = [[40,10],[30,20]], k = 1

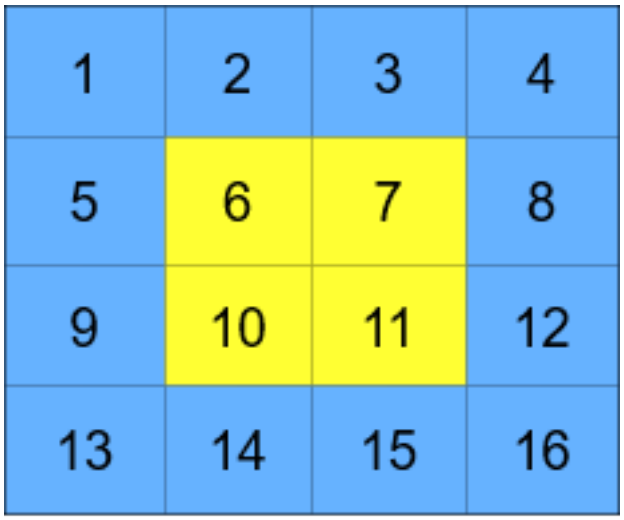
Output:

[[10,20],[40,30]]

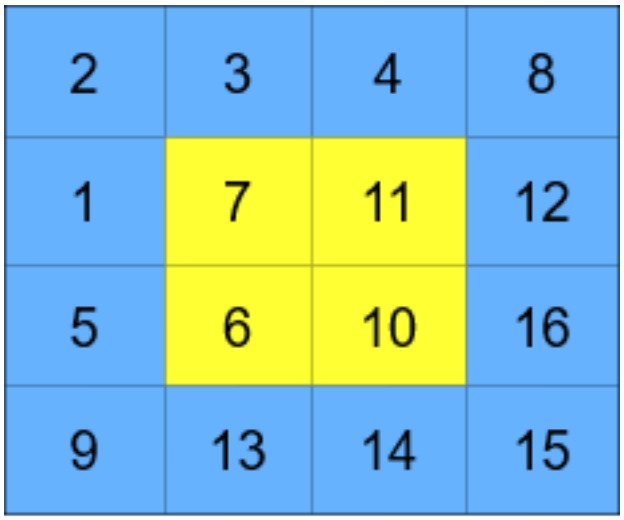
Explanation:

The figures above represent the grid at every state.

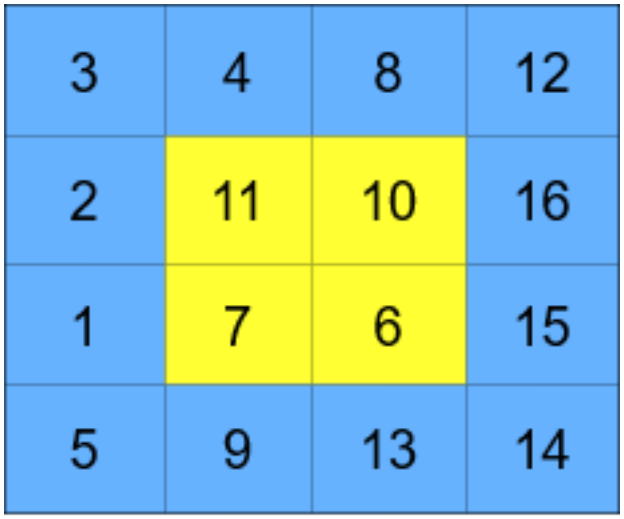
Example 2:



Before Any Rotations



After One Rotation



After Two Rotations

Input:

grid = [[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16]], k = 2

Output:

[[3,4,8,12],[2,11,10,16],[1,7,6,15],[5,9,13,14]]

Explanation:

The figures above represent the grid at every state.

Constraints:

m == grid.length

n == grid[i].length

2 <= m, n <= 50

Both

m

and

n

are

even

integers.

1 <= grid[i][j] <=

5000

1 <= k <= 10

9

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> rotateGrid(vector<vector<int>>& grid, int k) {

}
};
```

Java:

```
class Solution {
public int[][] rotateGrid(int[][] grid, int k) {

}
}
```

Python3:

```
class Solution:
def rotateGrid(self, grid: List[List[int]], k: int) -> List[List[int]]:
```

Python:

```
class Solution(object):
def rotateGrid(self, grid, k):
"""
:type grid: List[List[int]]
:type k: int
:rtype: List[List[int]]
"""
```

JavaScript:

```
/**
* @param {number[][]} grid
* @param {number} k
* @return {number[][]}
*/
var rotateGrid = function(grid, k) {

};
```

TypeScript:

```
function rotateGrid(grid: number[][], k: number): number[][] {

};
```

C#:

```
public class Solution {
public int[][] RotateGrid(int[][] grid, int k) {

}
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** rotateGrid(int** grid, int gridSize, int* gridColSize, int k, int*
returnSize, int** returnColumnSizes){

}
```

Go:

```
func rotateGrid(grid [][]int, k int) [][]int {

}
```

Kotlin:

```
class Solution {
fun rotateGrid(grid: Array<IntArray>, k: Int): Array<IntArray> {

}
}
```

Swift:

```
class Solution {
func rotateGrid(_ grid: [[Int]], _ k: Int) -> [[Int]] {

}
}
```

Rust:

```
impl Solution {
pub fn rotate_grid(grid: Vec<Vec<i32>>, k: i32) -> Vec<Vec<i32>> {

}
}
```

Ruby:

```
# @param {Integer[][]} grid
# @param {Integer} k
# @return {Integer[][]}
def rotate_grid(grid, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $grid
* @param Integer $k
* @return Integer[][]
*/
function rotateGrid($grid, $k) {

}
}
```

Scala:

```
object Solution {
def rotateGrid(grid: Array[Array[Int]], k: Int): Array[Array[Int]] = {

}
}
```

Racket:

```
(define/contract (rotate-grid grid k)
(-> (listof (listof exact-integer?)) exact-integer? (listof (listof
exact-integer?)))

)
```

Solutions

C++ Solution:

```
/*
* Problem: Cyclically Rotating a Grid
* Difficulty: Medium
* 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:
vector<vector<int>> rotateGrid(vector<vector<int>>& grid, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Cyclically Rotating a Grid
* Difficulty: Medium
* 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 int[][] rotateGrid(int[][] grid, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Cyclically Rotating a Grid
Difficulty: Medium
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 rotateGrid(self, grid: List[List[int]], k: int) -> List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def rotateGrid(self, grid, k):
"""
:type grid: List[List[int]]
:type k: int
:rtype: List[List[int]]
"""
```

JavaScript Solution:

```
/**
* Problem: Cyclically Rotating a Grid
* Difficulty: Medium
* 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[][]} grid
* @param {number} k
* @return {number[][]}
*/
var rotateGrid = function(grid, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Cyclically Rotating a Grid
* Difficulty: Medium
* 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 rotateGrid(grid: number[][], k: number): number[][] {

};
```

C# Solution:

```
/*
* Problem: Cyclically Rotating a Grid
* Difficulty: Medium
* 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 int[][] RotateGrid(int[][] grid, int k) {

}
}
```

C Solution:

```
/*
* Problem: Cyclically Rotating a Grid
* Difficulty: Medium
* 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
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** rotateGrid(int** grid, int gridSize, int* gridColSize, int k, int*
returnSize, int** returnColumnSizes){

}
```

Go Solution:

```
// Problem: Cyclically Rotating a Grid
// Difficulty: Medium
// 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 rotateGrid(grid [][]int, k int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun rotateGrid(grid: Array<IntArray>, k: Int): Array<IntArray> {

}
}
```

Swift Solution:

```
class Solution {
func rotateGrid(_ grid: [[Int]], _ k: Int) -> [[Int]] {
```

```
}
}
```

Rust Solution:

```
// Problem: Cyclically Rotating a Grid
// Difficulty: Medium
// 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 rotate_grid(grid: Vec<Vec<i32>>, k: i32) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} grid
# @param {Integer} k
# @return {Integer[][]}
def rotate_grid(grid, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $grid
* @param Integer $k
* @return Integer[][]
*/
function rotateGrid($grid, $k) {

}
}
```

Scala Solution:

```
object Solution {
def rotateGrid(grid: Array[Array[Int]], k: Int): Array[Array[Int]] = {

}
}
```

Racket Solution:

```
(define/contract (rotate-grid grid k)
(-> (listof (listof exact-integer?)) exact-integer? (listof (listof
exact-integer?)))

)
```