Problem 1260: Shift 2D Grid

Problem Information

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

Problem Description

Given a 2D

grid

of size

m x n

and an integer

k

. You need to shift the

grid

k

times.

In one shift operation:

Element at

grid[i][j]

moves to

grid[i][j + 1]

.

Element at

grid[i][n - 1]

moves to

grid[i + 1][0]

.

Element at

grid[m - 1][n - 1]

moves to

grid[0][0]

.

Return the

2D grid

after applying shift operation

k

times.

Example 1:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \rightarrow \begin{bmatrix} 9 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix}$$

Input:

grid

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

Output:

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

Example 2:

$$\begin{bmatrix} 3 & 8 & 1 & 9 \\ 19 & 7 & 2 & 5 \\ 4 & 6 & 11 & 10 \\ 12 & 0 & 21 & 13 \end{bmatrix} \rightarrow \begin{bmatrix} 13 & 3 & 8 & 1 \\ 9 & 19 & 7 & 2 \\ 5 & 4 & 6 & 11 \\ 10 & 12 & 0 & 21 \end{bmatrix} \rightarrow \begin{bmatrix} 21 & 13 & 3 & 8 \\ 1 & 9 & 19 & 7 \\ 2 & 5 & 4 & 6 \\ 11 & 10 & 12 & 0 \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} 0 & 21 & 13 & 3 \\ 8 & 1 & 9 & 19 \\ 7 & 2 & 5 & 4 \\ 6 & 11 & 10 & 12 \end{bmatrix} \rightarrow \begin{bmatrix} 12 & 0 & 21 & 13 \\ 3 & 8 & 1 & 9 \\ 19 & 7 & 2 & 5 \\ 4 & 6 & 11 & 10 \end{bmatrix}$$

Input:

grid

= [[3,8,1,9],[19,7,2,5],[4,6,11,10],[12,0,21,13]], k = 4

Output:

[[12,0,21,13],[3,8,1,9],[19,7,2,5],[4,6,11,10]]

Example 3:

Input:

grid

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

Output:

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

Constraints:

m == grid.length

n == grid[i].length

1 <= m <= 50

1 <= n <= 50

-1000 <= grid[i][j] <= 1000

0 <= k <= 100

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public List<List<Integer>> shiftGrid(int[][] grid, int k) {

}
}
```

Python3:

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

Python:

```
class Solution(object):
def shiftGrid(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 shiftGrid = function(grid, k) {

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public IList<IList<int>> ShiftGrid(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** shiftGrid(int** grid, int gridSize, int* gridColSize, int k, int*
returnSize, int** returnColumnSizes) {

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

```
}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
List<List<int>> shiftGrid(List<List<int>> grid, int k) {

}
}
```

Scala:

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

}
}
```

Elixir:

```
defmodule Solution do
@spec shift_grid(grid :: [[integer]], k :: integer) :: [[integer]]
def shift_grid(grid, k) do

end
end
```

Erlang:

```
-spec shift_grid(Grid :: [[integer()]], K :: integer()) -> [[integer()]].
shift_grid(Grid, K) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Shift 2D Grid
* 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:
vector<vector<int>> shiftGrid(vector<vector<int>>& grid, int k) {

}
```

```
};
```

Java Solution:

```
/**
* Problem: Shift 2D Grid
* 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 List<List<Integer>> shiftGrid(int[][] grid, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Shift 2D Grid
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 shiftGrid(self, grid: List[List[int]], k: int) -> List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def shiftGrid(self, grid, k):
```

```
"""
:type grid: List[List[int]]
:type k: int
:rtype: List[List[int]]
"""
```

JavaScript Solution:

```
/**
* Problem: Shift 2D Grid
* 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[][]} grid
* @param {number} k
* @return {number[][]}
*/
var shiftGrid = function(grid, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Shift 2D Grid
* 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 shiftGrid(grid: number[][], k: number): number[][] {
```

```
};
```

C# Solution:

```
/*
* Problem: Shift 2D Grid
* 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 IList<IList<int>> ShiftGrid(int[][] grid, int k) {

}
}
```

C Solution:

```
/*
* Problem: Shift 2D Grid
* 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
*/

/**
* 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** shiftGrid(int** grid, int gridSize, int* gridColSize, int k, int*
returnSize, int** returnColumnSizes) {
```

```
}
```

Go Solution:

```
// Problem: Shift 2D Grid
// 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 shiftGrid(grid [][]int, k int) [][]int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Shift 2D Grid
// 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 shift_grid(grid: Vec<Vec<i32>>, k: i32) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> shiftGrid(List<List<int>> grid, int k) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec shift_grid(grid :: [[integer]], k :: integer) :: [[integer]]
def shift_grid(grid, k) do

end
end
```

Erlang Solution:

```
-spec shift_grid(Grid :: [[integer()]], K :: integer()) -> [[integer()]].
shift_grid(Grid, K) ->
.
```

Racket Solution:

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