Problem 2435: Paths in Matrix Whose Sum Is Divisible by K

Problem Information

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

Problem Description

You are given a

0-indexed

m x n

integer matrix

grid

and an integer

k

. You are currently at position

(0, 0)

and you want to reach position

(m - 1, n - 1)

moving only

down

or

right

.

Return

the number of paths where the sum of the elements on the path is divisible by

k

. Since the answer may be very large, return it

modulo

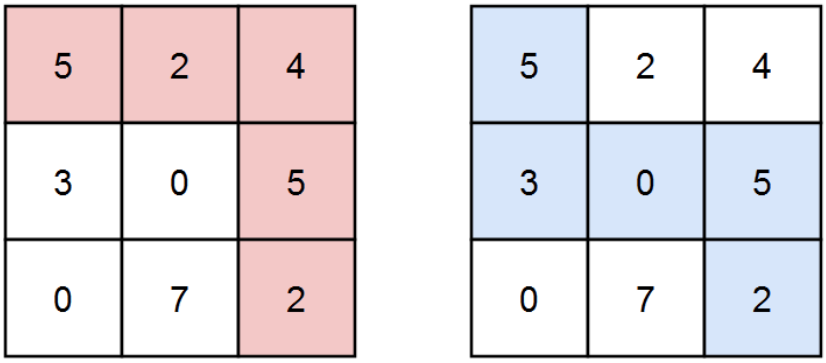
10

9

+ 7

.

Example 1:



Input:

grid = [[5,2,4],[3,0,5],[0,7,2]], k = 3

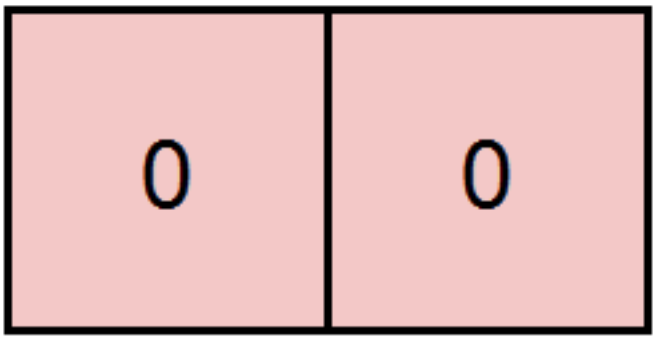
Output:

2

Explanation:

There are two paths where the sum of the elements on the path is divisible by k. The first path highlighted in red has a sum of 5 + 2 + 4 + 5 + 2 = 18 which is divisible by 3. The second path highlighted in blue has a sum of 5 + 3 + 0 + 5 + 2 = 15 which is divisible by 3.

Example 2:



Input:

grid = [[0,0]], k = 5

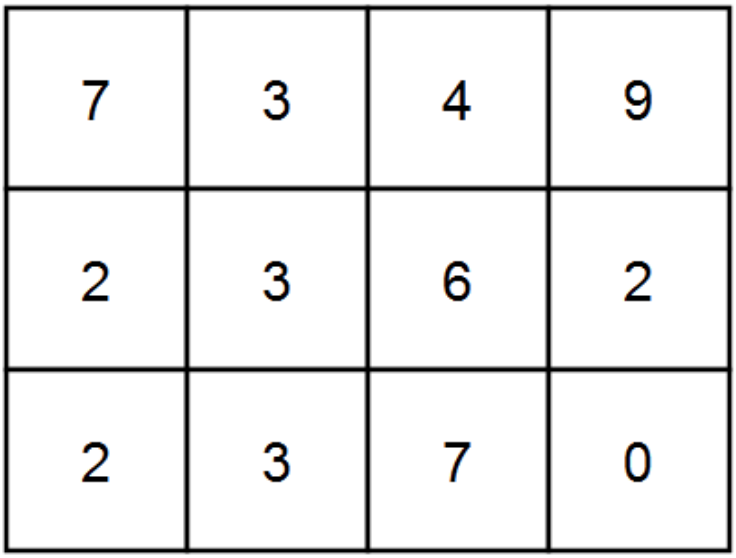
Output:

1

Explanation:

The path highlighted in red has a sum of 0 + 0 = 0 which is divisible by 5.

Example 3:



Input:

grid = [[7,3,4,9],[2,3,6,2],[2,3,7,0]], k = 1

Output:

10

Explanation:

Every integer is divisible by 1 so the sum of the elements on every possible path is divisible by k.

Constraints:

m == grid.length

n == grid[i].length

1 <= m, n <= 5 * 10

4

1 <= m * n <= 5 * 10

4

0 <= grid[i][j] <= 100

1 <= k <= 50

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

```
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int numberOfPaths(int** grid, int gridSize, int* gridColSize, int k) {

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
```

```
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (number-of-paths grid k)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Paths in Matrix Whose Sum Is Divisible by K
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Paths in Matrix Whose Sum Is Divisible by K
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Paths in Matrix Whose Sum Is Divisible by K
Difficulty: Hard
Tags: array, dp
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def numberOfPaths(self, grid: List[List[int]], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Paths in Matrix Whose Sum Is Divisible by K
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Paths in Matrix Whose Sum Is Divisible by K
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function numberOfPaths(grid: number[][], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Paths in Matrix Whose Sum Is Divisible by K
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

C Solution:

```
/*
* Problem: Paths in Matrix Whose Sum Is Divisible by K
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int numberOfPaths(int** grid, int gridSize, int* gridColSize, int k) {

}
```

Go Solution:

```
// Problem: Paths in Matrix Whose Sum Is Divisible by K
// Difficulty: Hard
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func numberOfPaths(grid [][]int, k int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Paths in Matrix Whose Sum Is Divisible by K
// Difficulty: Hard
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

```
}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (number-of-paths grid k)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```