Problem 2906: Construct Product Matrix

Problem Information

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

Problem Description

Given a

0-indexed

2D integer matrix

grid

of size

n * m

, we define a

0-indexed

2D matrix

p

of size

n * m

as the

product

matrix of

grid

if the following condition is met:

Each element

p[i][j]

is calculated as the product of all elements in

grid

except for the element

grid[i][j]

. This product is then taken modulo

12345

.

Return

the product matrix of

grid

.

Example 1:

Input:

grid = [[1,2],[3,4]]

Output:

[[24,12],[8,6]]

Explanation:

p[0][0] = grid[0][1] * grid[1][0] * grid[1][1] = 2 * 3 * 4 = 24 p[0][1] = grid[0][0] * grid[1][0] * grid[1][1] = 1 * 3 * 4 = 12 p[1][0] = grid[0][0] * grid[0][1] * grid[1][1] = 1 * 2 * 4 = 8 p[1][1] = grid[0][0] * grid[0][1] * grid[1][0] = 1 * 2 * 3 = 6 So the answer is [[24,12],[8,6]].

Example 2:

Input:

grid = [[12345],[2],[1]]

Output:

[[2],[0],[0]]

Explanation:

p[0][0] = grid[0][1] * grid[0][2] = 2 * 1 = 2. p[0][1] = grid[0][0] * grid[0][2] = 12345 * 1 = 12345. 12345 % 12345 = 0. So p[0][1] = 0. p[0][2] = grid[0][0] * grid[0][1] = 12345 * 2 = 24690. 24690 % 12345 = 0. So p[0][2] = 0. So the answer is [[2],[0],[0]].

Constraints:

1 <= n == grid.length <= 10

5

1 <= m == grid[i].length <= 10

5

2 <= n * m <= 10

5

1 <= grid[i][j] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> constructProductMatrix(vector<vector<int>>& grid) {

}
};
```

Java:

```
class Solution {
public int[][] constructProductMatrix(int[][] grid) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[][]} grid
```

```
* @return {number[][]}
*/
var constructProductMatrix = function(grid) {

};
```

TypeScript:

```
function constructProductMatrix(grid: number[][]): number[][] {

};
```

C#:

```
public class Solution {
public int[][] ConstructProductMatrix(int[][] grid) {

}
}
```

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** constructProductMatrix(int** grid, int gridSize, int* gridColSize, int*
returnSize, int** returnColumnSizes) {

}
```

Go:

```
func constructProductMatrix(grid [][]int) [][]int {

}
```

Kotlin:

```
class Solution {
fun constructProductMatrix(grid: Array<IntArray>): Array<IntArray> {

}
}
```

Swift:

```
class Solution {
func constructProductMatrix(_ grid: [[Int]]) -> [[Int]] {

}
}
```

Rust:

```
impl Solution {
pub fn construct_product_matrix(grid: Vec<Vec<i32>>) -> Vec<Vec<i32>> {

}
}
```

Ruby:

```
# @param {Integer[][]} grid
# @return {Integer[][]}
def construct_product_matrix(grid)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $grid
* @return Integer[][]
*/
function constructProductMatrix($grid) {

}
}
```

Dart:

```
class Solution {
List<List<int>> constructProductMatrix(List<List<int>> grid) {

}
}
```

Scala:

```
object Solution {
def constructProductMatrix(grid: Array[Array[Int]]): Array[Array[Int]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec construct_product_matrix(grid :: [[integer]]) :: [[integer]]
def construct_product_matrix(grid) do

end
end
```

Erlang:

```
-spec construct_product_matrix(Grid :: [[integer()]]) -> [[integer()]].
construct_product_matrix(Grid) ->
.
```

Racket:

```
(define/contract (construct-product-matrix grid)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Construct Product Matrix
* 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>> constructProductMatrix(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Construct Product Matrix
* 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[][] constructProductMatrix(int[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Construct Product Matrix
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 constructProductMatrix(self, grid: List[List[int]]) -> List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Construct Product Matrix
* 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
* @return {number[][]}
*/
var constructProductMatrix = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Construct Product Matrix
* 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 constructProductMatrix(grid: number[][]): number[][] {

};
```

C# Solution:

```
/*
* Problem: Construct Product Matrix
* 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[][] ConstructProductMatrix(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Construct Product Matrix
* 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** constructProductMatrix(int** grid, int gridSize, int* gridColSize, int*
returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Construct Product Matrix
// 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 constructProductMatrix(grid [][]int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun constructProductMatrix(grid: Array<IntArray>): Array<IntArray> {

}
}
```

Swift Solution:

```
class Solution {
func constructProductMatrix(_ grid: [[Int]]) -> [[Int]] {

}
```

```
}
```

Rust Solution:

```
// Problem: Construct Product Matrix
// 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 construct_product_matrix(grid: Vec<Vec<i32>>) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} grid
# @return {Integer[][]}
def construct_product_matrix(grid)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $grid
* @return Integer[][]
*/
function constructProductMatrix($grid) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> constructProductMatrix(List<List<int>> grid) {

}
}
```

Scala Solution:

```
object Solution {
def constructProductMatrix(grid: Array[Array[Int]]): Array[Array[Int]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec construct_product_matrix(grid :: [[integer]]) :: [[integer]]
def construct_product_matrix(grid) do

end
end
```

Erlang Solution:

```
-spec construct_product_matrix(Grid :: [[integer()]]) -> [[integer()]].
construct_product_matrix(Grid) ->
.
```

Racket Solution:

```
(define/contract (construct-product-matrix grid)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)
```