Problem 1594: Maximum Non Negative Product in a Matrix

Problem Information

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

Problem Description

You are given a

m x n

matrix

grid

. Initially, you are located at the top-left corner

(0, 0)

, and in each step, you can only

move right or down

in the matrix.

Among all possible paths starting from the top-left corner

(0, 0)

and ending in the bottom-right corner

(m - 1, n - 1)

, find the path with the

maximum non-negative product

. The product of a path is the product of all integers in the grid cells visited along the path.

Return the

maximum non-negative product

modulo

10

9

+ 7

.

If the maximum product is

negative

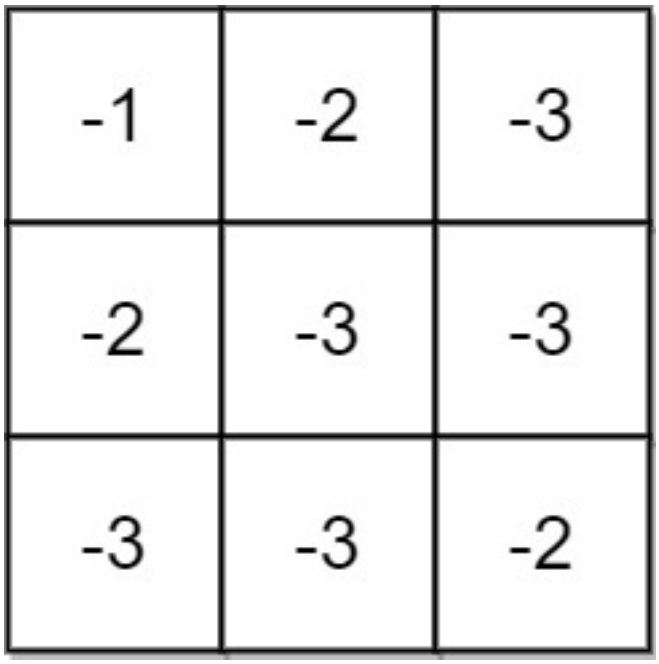
, return

-1

.

Notice that the modulo is performed after getting the maximum product.

Example 1:



Input:

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

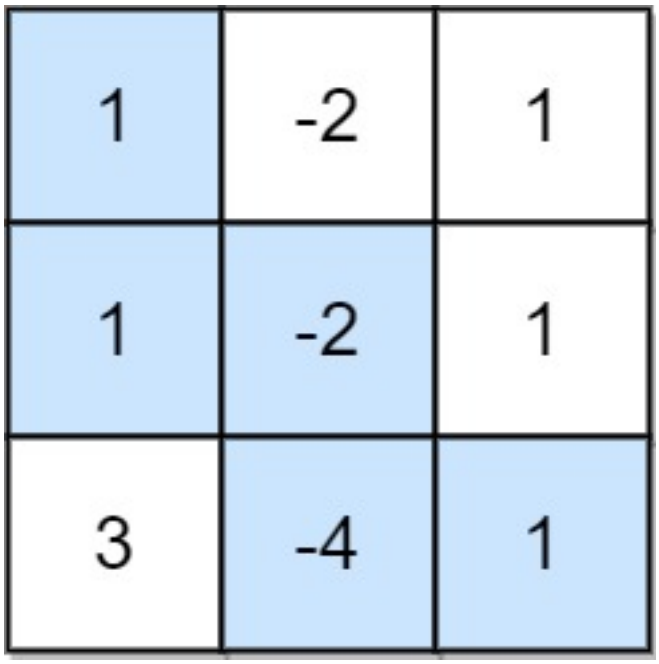
Output:

-1

Explanation:

It is not possible to get non-negative product in the path from (0, 0) to (2, 2), so return -1.

Example 2:



Input:

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

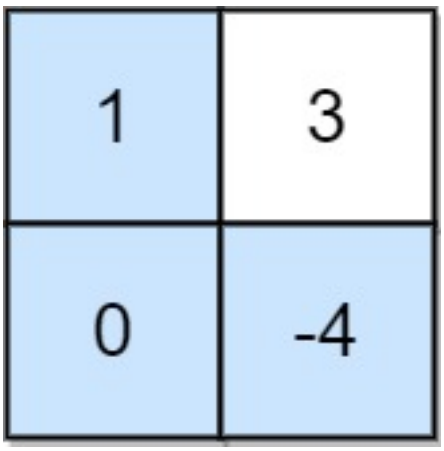
Output:

8

Explanation:

Maximum non-negative product is shown (1 * 1 * -2 * -4 * 1 = 8).

Example 3:



Input:

grid = [[1,3],[0,-4]]

Output:

0

Explanation:

Maximum non-negative product is shown (1 * 0 * -4 = 0).

Constraints:

m == grid.length

n == grid[i].length

1 <= m, n <= 15

-4 <= grid[i][j] <= 4

Code Snippets

C++:

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

}
};
```

Java:

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

}
```

```
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int maxProductPath(int** grid, int gridSize, int* gridColSize) {

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (max-product-path grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Non Negative Product in a Matrix
* Difficulty: Medium
* 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 maxProductPath(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Non Negative Product in a Matrix
* Difficulty: Medium
* 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 maxProductPath(int[][] grid) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Maximum Non Negative Product in a Matrix
Difficulty: Medium
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 maxProductPath(self, grid: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Non Negative Product in a Matrix
* Difficulty: Medium
* 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
* @return {number}
*/
var maxProductPath = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Non Negative Product in a Matrix
* Difficulty: Medium
* 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 maxProductPath(grid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Non Negative Product in a Matrix
* Difficulty: Medium
* 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 MaxProductPath(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Non Negative Product in a Matrix
* Difficulty: Medium
* 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 maxProductPath(int** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```
// Problem: Maximum Non Negative Product in a Matrix
// Difficulty: Medium
// 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 maxProductPath(grid [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

```
}
}
```

Rust Solution:

```
// Problem: Maximum Non Negative Product in a Matrix
// Difficulty: Medium
// 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 max_product_path(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (max-product-path grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```