Problem 1970: Last Day Where You Can Still Cross

Problem Information

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

Problem Description

There is a

1-based

binary matrix where

0

represents land and

1

represents water. You are given integers

row

and

col

representing the number of rows and columns in the matrix, respectively.

Initially on day

0

, the

entire

matrix is

land

. However, each day a new cell becomes flooded with

water

. You are given a

1-based

2D array

cells

, where

cells[i] = [r

i

, c

i

]

represents that on the

i

th

day, the cell on the

r

i

th

row and

c

i

th

column (

1-based

coordinates) will be covered with

water

(i.e., changed to

1

).

You want to find the

last

day that it is possible to walk from the

top

to the

bottom

by only walking on land cells. You can start from

any

cell in the top row and end at

any

cell in the bottom row. You can only travel in the

four

cardinal directions (left, right, up, and down).

Return

the

last

day where it is possible to walk from the

top

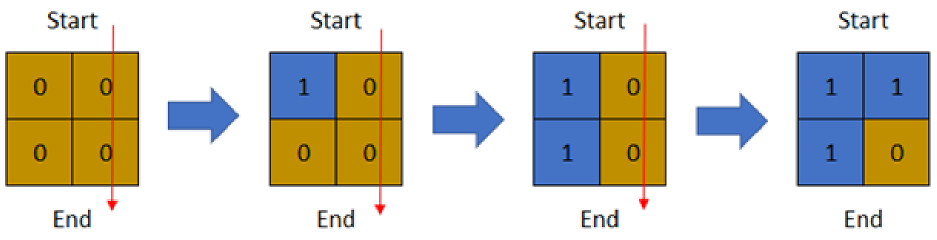
to the

bottom

by only walking on land cells

.

Example 1:



Input:

row = 2, col = 2, cells = [[1,1],[2,1],[1,2],[2,2]]

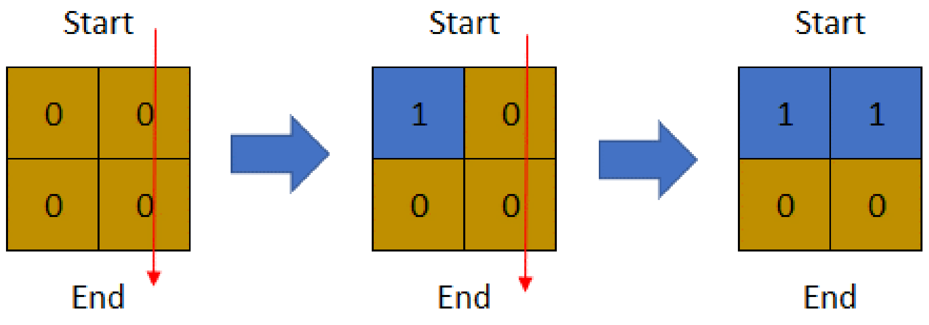
Output:

2

Explanation:

The above image depicts how the matrix changes each day starting from day 0. The last day where it is possible to cross from top to bottom is on day 2.

Example 2:



Input:

row = 2, col = 2, cells = [[1,1],[1,2],[2,1],[2,2]]

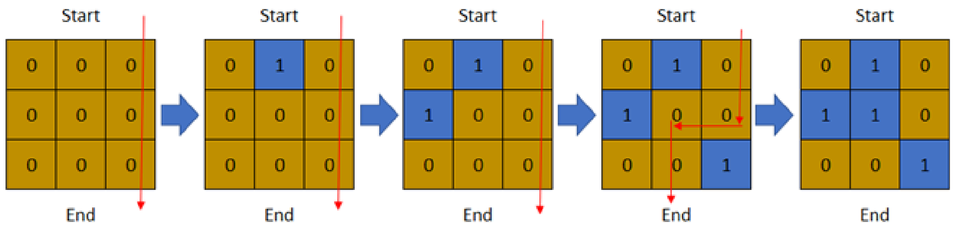
Output:

1

Explanation:

The above image depicts how the matrix changes each day starting from day 0. The last day where it is possible to cross from top to bottom is on day 1.

Example 3:



Input:

row = 3, col = 3, cells = [[1,2],[2,1],[3,3],[2,2],[1,1],[1,3],[2,3],[3,2],[3,1]]

Output:

3

Explanation:

The above image depicts how the matrix changes each day starting from day 0. The last day where it is possible to cross from top to bottom is on day 3.

Constraints:

2 <= row, col <= 2 * 10

4

4 <= row * col <= 2 * 10

4

cells.length == row * col

1 <= r

i

<= row

1 <= c

i

<= col

All the values of

cells

are

unique

.

Code Snippets

C++:

```
class Solution {
public:
int latestDayToCross(int row, int col, vector<vector<int>>& cells) {

}
};
```

Java:

```
class Solution {
public int latestDayToCross(int row, int col, int[][] cells) {

}
}
```

Python3:

```
class Solution:
def latestDayToCross(self, row: int, col: int, cells: List[List[int]]) ->
int:
```

Python:

```
class Solution(object):
def latestDayToCross(self, row, col, cells):
"""
:type row: int
:type col: int
:type cells: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} row
* @param {number} col
* @param {number[][]} cells
* @return {number}
*/
var latestDayToCross = function(row, col, cells) {

};
```

TypeScript:

```
function latestDayToCross(row: number, col: number, cells: number[][]):
number {

};
```

C#:

```
public class Solution {
public int LatestDayToCross(int row, int col, int[][] cells) {

}
}
```

C:

```
int latestDayToCross(int row, int col, int** cells, int cellsSize, int*
cellsColSize) {

}
```

Go:

```
func latestDayToCross(row int, col int, cells [][]int) int {

}
```

Kotlin:

```
class Solution {
fun latestDayToCross(row: Int, col: Int, cells: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func latestDayToCross(_ row: Int, _ col: Int, _ cells: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn latest_day_to_cross(row: i32, col: i32, cells: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} row
# @param {Integer} col
# @param {Integer[][]} cells
# @return {Integer}
def latest_day_to_cross(row, col, cells)
```

```
end
```

PHP:

```
class Solution {

/**
* @param Integer $row
* @param Integer $col
* @param Integer[][] $cells
* @return Integer
*/
function latestDayToCross($row, $col, $cells) {

}
}
```

Dart:

```
class Solution {
int latestDayToCross(int row, int col, List<List<int>> cells) {

}
}
```

Scala:

```
object Solution {
def latestDayToCross(row: Int, col: Int, cells: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec latest_day_to_cross(row :: integer, col :: integer, cells ::
[[integer]]) :: integer
def latest_day_to_cross(row, col, cells) do

end
end
```

Erlang:

```
-spec latest_day_to_cross(Row :: integer(), Col :: integer(), Cells ::
[[integer()]]) -> integer().
latest_day_to_cross(Row, Col, Cells) ->
.
```

Racket:

```
(define/contract (latest-day-to-cross row col cells)
(-> exact-integer? exact-integer? (listof (listof exact-integer?))
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Last Day Where You Can Still Cross
* Difficulty: Hard
* Tags: array, graph, search
*
* 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 latestDayToCross(int row, int col, vector<vector<int>>& cells) {

}
};
```

Java Solution:

```
/**
* Problem: Last Day Where You Can Still Cross
* Difficulty: Hard
* Tags: array, graph, search
```

```
*
* 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 latestDayToCross(int row, int col, int[][] cells) {

}
}
```

Python3 Solution:

```
"""
Problem: Last Day Where You Can Still Cross
Difficulty: Hard
Tags: array, graph, search

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 latestDayToCross(self, row: int, col: int, cells: List[List[int]]) ->
int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def latestDayToCross(self, row, col, cells):
"""
:type row: int
:type col: int
:type cells: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Last Day Where You Can Still Cross
* Difficulty: Hard
* Tags: array, graph, search
*
* 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} row
* @param {number} col
* @param {number[][]} cells
* @return {number}
*/
var latestDayToCross = function(row, col, cells) {

};
```

TypeScript Solution:

```
/**
* Problem: Last Day Where You Can Still Cross
* Difficulty: Hard
* Tags: array, graph, search
*
* 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 latestDayToCross(row: number, col: number, cells: number[][]):
number {

};
```

C# Solution:

```
/*
* Problem: Last Day Where You Can Still Cross
* Difficulty: Hard
* Tags: array, graph, search
*
* 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 LatestDayToCross(int row, int col, int[][] cells) {

}
}
```

C Solution:

```
/*
* Problem: Last Day Where You Can Still Cross
* Difficulty: Hard
* Tags: array, graph, search
*
* 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
*/

int latestDayToCross(int row, int col, int** cells, int cellsSize, int*
cellsColSize) {

}
```

Go Solution:

```
// Problem: Last Day Where You Can Still Cross
// Difficulty: Hard
// Tags: array, graph, search
//
// 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 latestDayToCross(row int, col int, cells [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun latestDayToCross(row: Int, col: Int, cells: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func latestDayToCross(_ row: Int, _ col: Int, _ cells: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Last Day Where You Can Still Cross
// Difficulty: Hard
// Tags: array, graph, search
//
// 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 latest_day_to_cross(row: i32, col: i32, cells: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} row
# @param {Integer} col
# @param {Integer[][]} cells
# @return {Integer}
def latest_day_to_cross(row, col, cells)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $row
* @param Integer $col
* @param Integer[][] $cells
* @return Integer
*/
function latestDayToCross($row, $col, $cells) {

}
}
```

Dart Solution:

```
class Solution {
int latestDayToCross(int row, int col, List<List<int>> cells) {

}
}
```

Scala Solution:

```
object Solution {
def latestDayToCross(row: Int, col: Int, cells: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec latest_day_to_cross(row :: integer, col :: integer, cells ::
[[integer]]) :: integer
```

```
def latest_day_to_cross(row, col, cells) do

end
end
```

Erlang Solution:

```
-spec latest_day_to_cross(Row :: integer(), Col :: integer(), Cells ::
[[integer()]]) -> integer().
latest_day_to_cross(Row, Col, Cells) ->
.
```

Racket Solution:

```
(define/contract (latest-day-to-cross row col cells)
(-> exact-integer? exact-integer? (listof (listof exact-integer?))
exact-integer?)
)
```