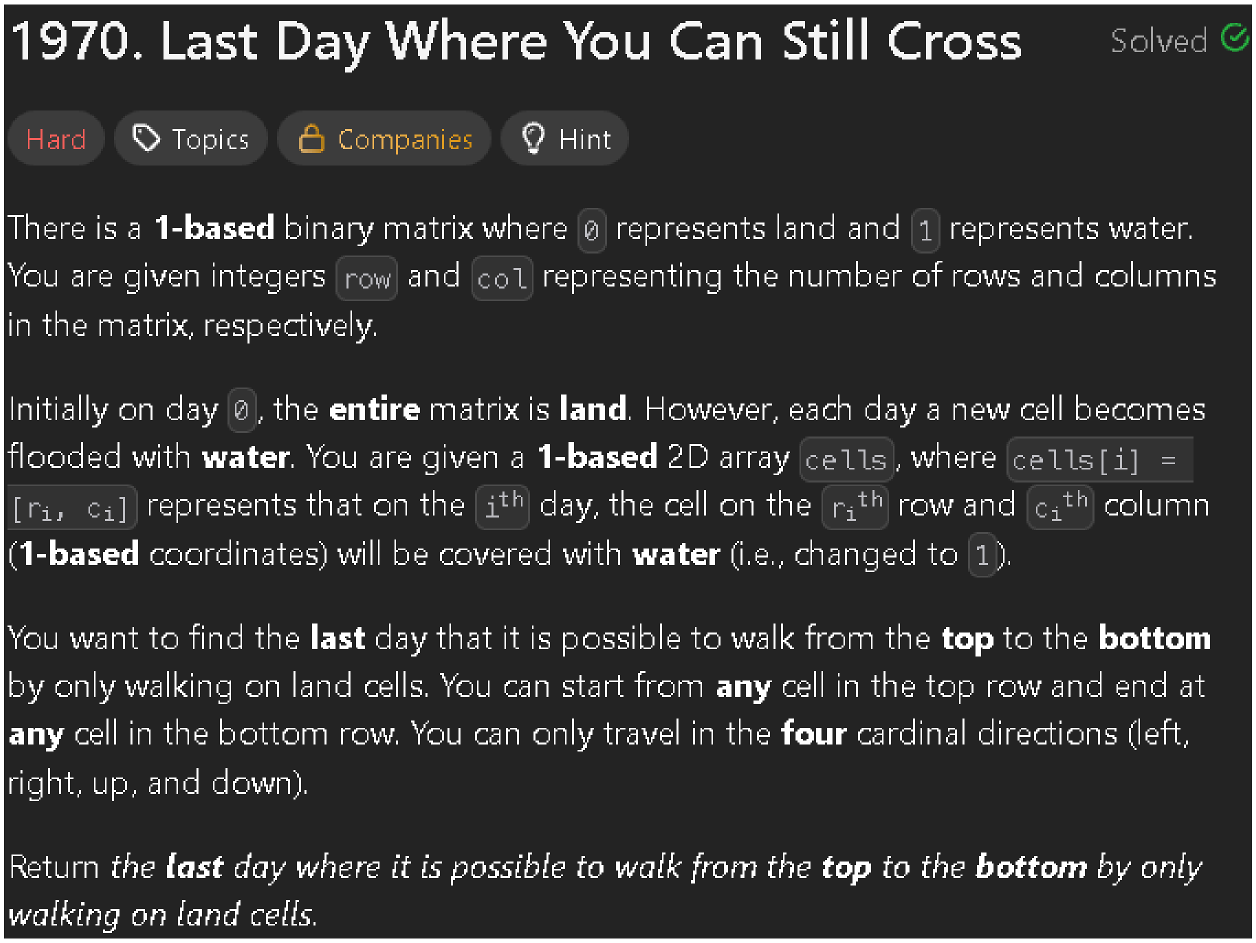


1970. Last Day Where You Can Still Cross

Solved

Hard · Topics · Companies · Hint

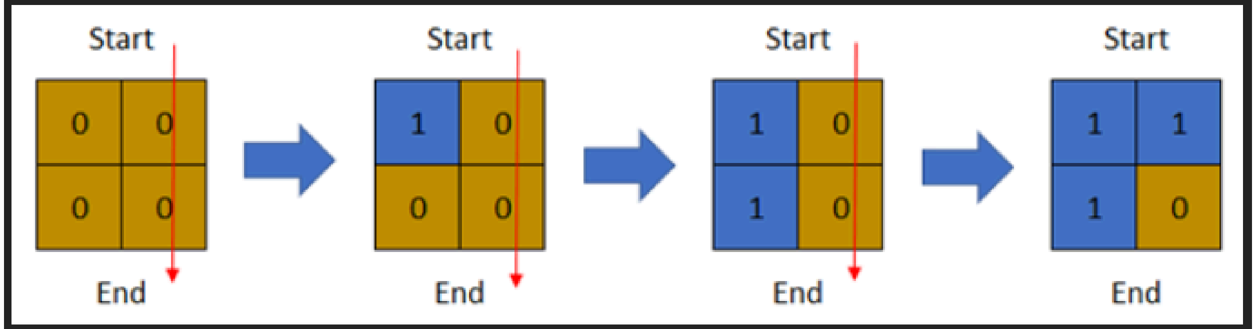
There is a **1-based** binary matrix where `0` represents land and `1` represents water. You are given integers `row` and `col` representing the number of rows and columns in the matrix, respectively.

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] = [ri, ci]` 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).

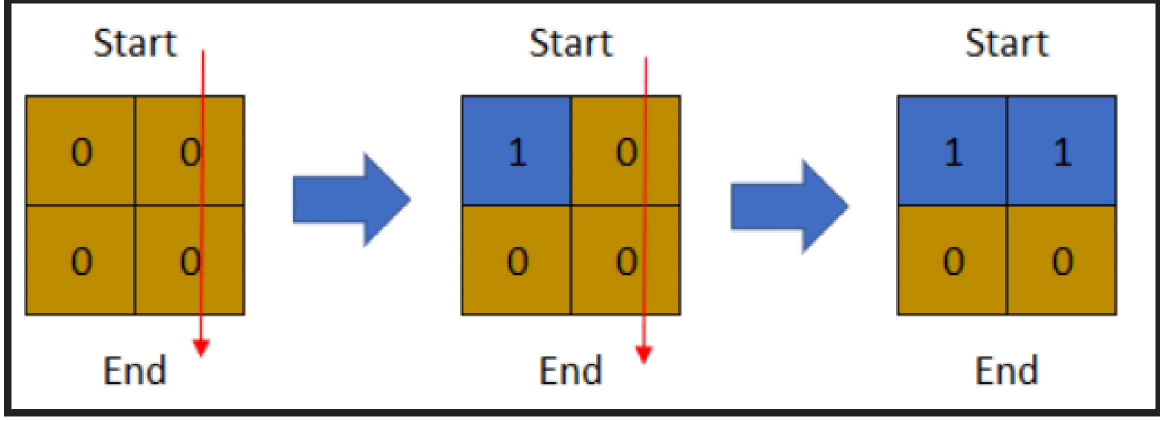
Return *the* ***last*** *day where it is possible to walk from the* ***top*** *to the* ***bottom*** *by only walking on land cells.*

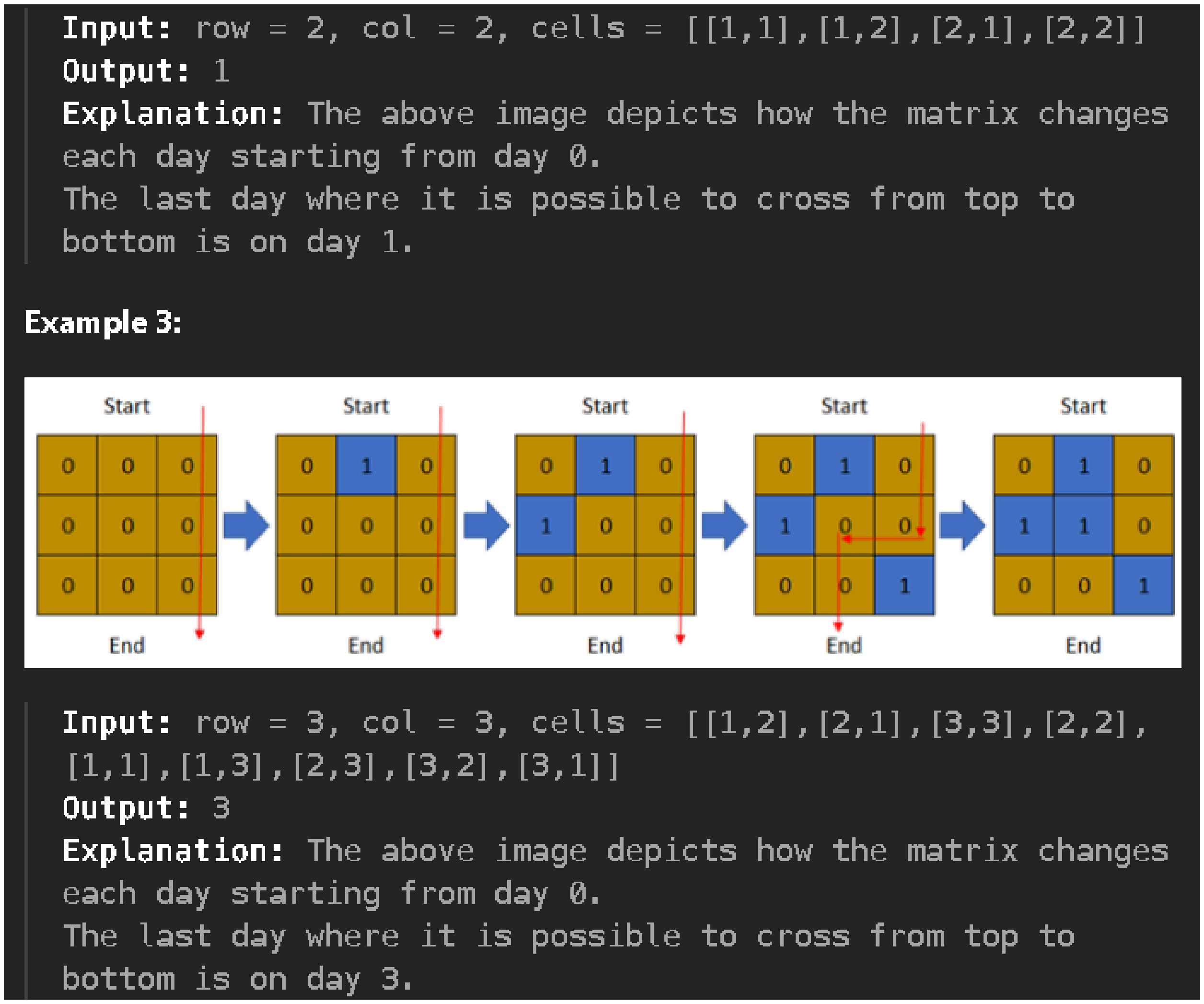
Example 1:



```
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.
```

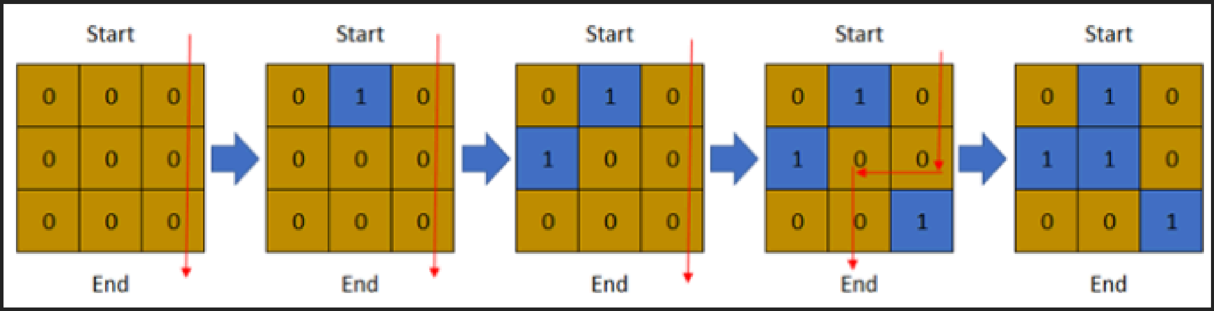
Example 2:



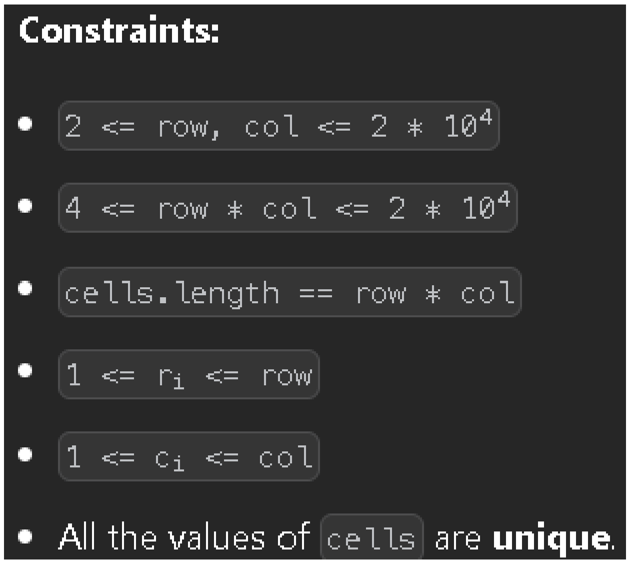


```
Input: row = 2, col = 2, cells = [[1,1],[1,2],[2,1],[2,2]]
Output: 1
Explanation: The above image depicts how the matrix changes
each day starting from day 0.
The last day where it is possible to cross from top to
bottom is on day 1.
```

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**.

Python:

class Solution:

```
def latestDayToCross(self, row: int, col: int, cells: List[List[int]]) -> int:
    n = row * col
    top, bottom = n, n + 1

    parent = list(range(n + 2))
    rank = [0] * (n + 2)
    grid = [[False] * col for _ in range(row)]

    def find(x):
        if parent[x] != x:
            parent[x] = find(parent[x])
        return parent[x]

    def union(a, b):
        a, b = find(a), find(b)
        if a == b:
            return
        if rank[a] < rank[b]:
            parent[a] = b
        else:
            parent[b] = a
            if rank[a] == rank[b]:
                rank[a] += 1

    dr = [1, -1, 0, 0]
    dc = [0, 0, 1, -1]

    for d in range(n - 1, -1, -1):
        r, c = cells[d][0] - 1, cells[d][1] - 1
        grid[r][c] = True
        idx = r * col + c

        if r == 0:
            union(idx, top)
        if r == row - 1:
            union(idx, bottom)

        for k in range(4):
            nr, nc = r + dr[k], c + dc[k]
            if 0 <= nr < row and 0 <= nc < col and grid[nr][nc]:
                union(idx, nr * col + nc)

        if find(top) == find(bottom):
            return d
        return 0
```

JavaScript:

```
/**
 * @param {number} row
 * @param {number} col
 * @param {number[][]} cells
 * @return {number}
 */
var latestDayToCross = function(row, col, cells) {
    const n = row * col;
    const top = n, bottom = n + 1;

    const parent = Array(n + 2).fill(0).map((_, i) => i);
    const rank = Array(n + 2).fill(0);
    const grid = Array.from({ length: row }, () => Array(col).fill(false));

    const find = (x) => {
        if (parent[x] !== x) parent[x] = find(parent[x]);
        return parent[x];
    };

    const union = (a, b) => {
        a = find(a);
        b = find(b);
        if (a === b) return;
        if (rank[a] < rank[b]) parent[a] = b;
        else {
            parent[b] = a;
            if (rank[a] === rank[b]) rank[a]++;
        }
    };

    const dr = [1, -1, 0, 0];
    const dc = [0, 0, 1, -1];

    for (let d = n - 1; d >= 0; d--) {
        const r = cells[d][0] - 1;
        const c = cells[d][1] - 1;
        grid[r][c] = true;
        const id = r * col + c;

        if (r === 0) union(id, top);
        if (r === row - 1) union(id, bottom);
        for (let k = 0; k < 4; k++) {
            const nr = r + dr[k];
            const nc = c + dc[k];
            if (nr >= 0 && nr < row && nc >= 0 && nc < col && grid[nr][nc]) {
                union(id, nr * col + nc);
            }
        }

        if (find(top) === find(bottom)) return d;
    }
    return 0;
};
```

Java:

```
class Solution {
    public int latestDayToCross(int row, int col, int[][] cells) {
        int n = row * col;
        int top = n, bottom = n + 1;

        int[] parent = new int[n + 2];
        int[] rank = new int[n + 2];
        boolean[][] grid = new boolean[row][col];

        for (int i = 0; i < n + 2; i++) parent[i] = i;

        int[] dr = {1, -1, 0, 0};
        int[] dc = {0, 0, 1, -1};

        for (int d = n - 1; d >= 0; d--) {
            int r = cells[d][0] - 1;
            int c = cells[d][1] - 1;
            grid[r][c] = true;
            int id = r * col + c;

            if (r == 0) union(id, top, parent, rank);
            if (r == row - 1) union(id, bottom, parent, rank);

            for (int k = 0; k < 4; k++) {
                int nr = r + dr[k];
                int nc = c + dc[k];
                if (nr >= 0 && nr < row && nc >= 0 && nc < col && grid[nr][nc]) {
                    union(id, nr * col + nc, parent, rank);
                }
            }

            if (find(top, parent) == find(bottom, parent)) return d;
        }
        return 0;
    }

    private int find(int x, int[] parent) {
        if (parent[x] != x) parent[x] = find(parent[x], parent);
        return parent[x];
    }

    private void union(int a, int b, int[] parent, int[] rank) {
        a = find(a, parent);
        b = find(b, parent);
        if (a == b) return;
        if (rank[a] < rank[b]) {
            parent[a] = b;
        } else {
            parent[b] = a;
            if (rank[a] == rank[b]) rank[a]++;
        }
    }
}
```