Rotting Oranges

You are given an m x n grid where each cell can have one of three values:

0 representing an empty cell,
1 representing a fresh orange, or
2 representing a rotten orange.
Every minute, any fresh orange that is 4-directionally adjacent to a rotten orange becomes rotten.

Return the minimum number of minutes that must elapse until no cell has a fresh orange.
If this is impossible, return -1.

Input: grid = [[2,1,1],[1,1,0],[0,1,1]]
Output: 4

Input: grid = [[2,1,1],[0,1,1],[1,0,1]]
Output: -1
Explanation: The orange in the bottom left corner (row 2, column 0) is never rotten, because rotting only happens 4-directionally.

Input: grid = [[1]]
Output: -1
Explanation: Since the orange can never rotate.

Input: grid = [[0,0,0,0,0]]
Output: 0
Explanation: Since there are no oranges to rotate.

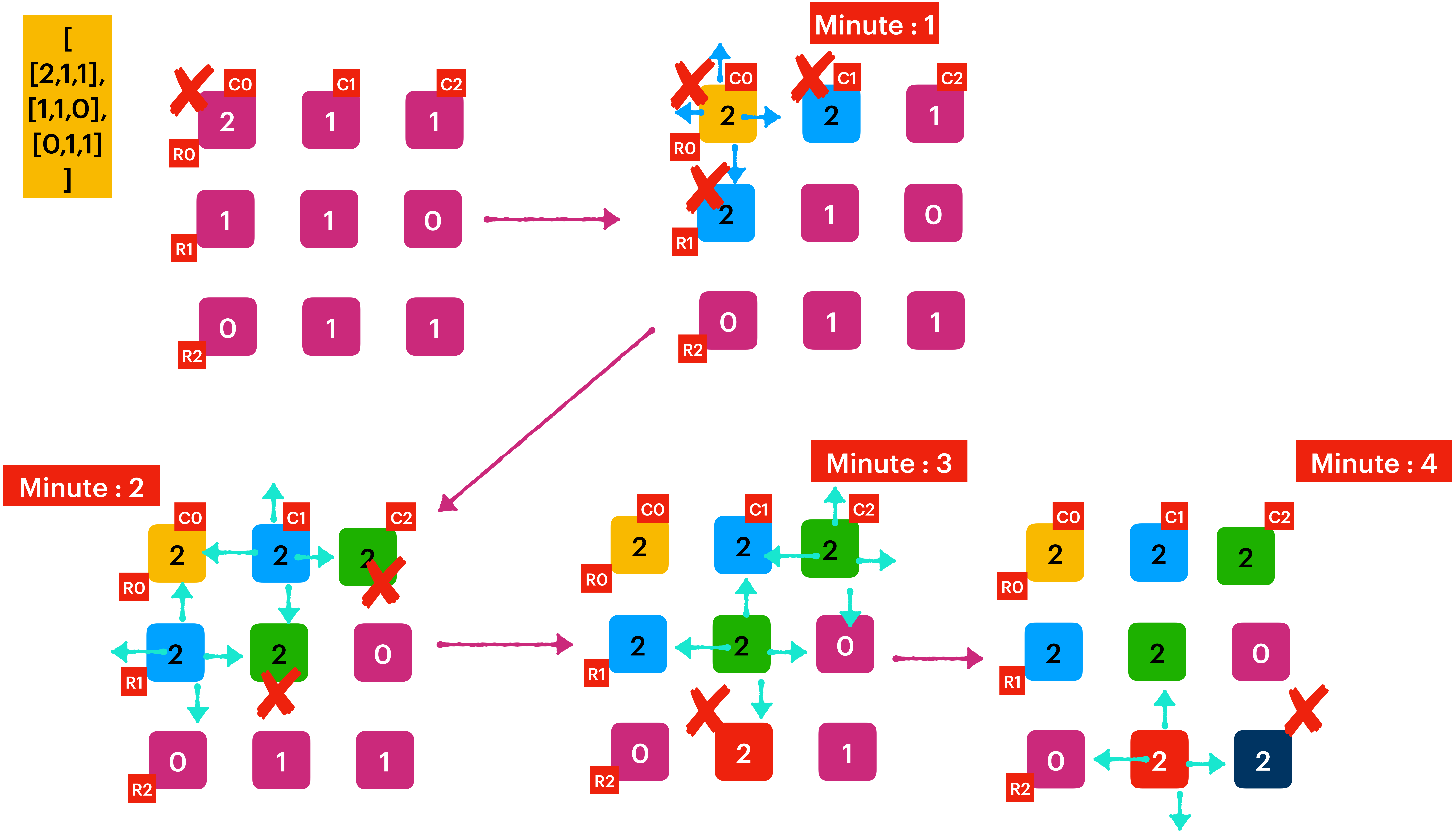
Input: grid = [[0]]
Output: 0
Explanation: Since there are already no fresh oranges at minute 0, the answer is just 0.

Input: grid = [[0,2]]
Output: 0
Explanation: Since there are already no fresh oranges at minute 0, the answer is just 0.

Constraints:

m == grid.length
n == grid[i].length
1 <= m, n <= 10
grid[i][j] is 0, 1, or 2.

[
[2,1,1],
[1,1,0],
[0,1,1]
]



	C0	C1	C2
R0	2	1	1
R1	1	1	0
R2	0	1	1

Minute : 1

	C0	C1	C2
R0	2	2	1
R1	2	1	0
R2	0	1	1

Minute : 2

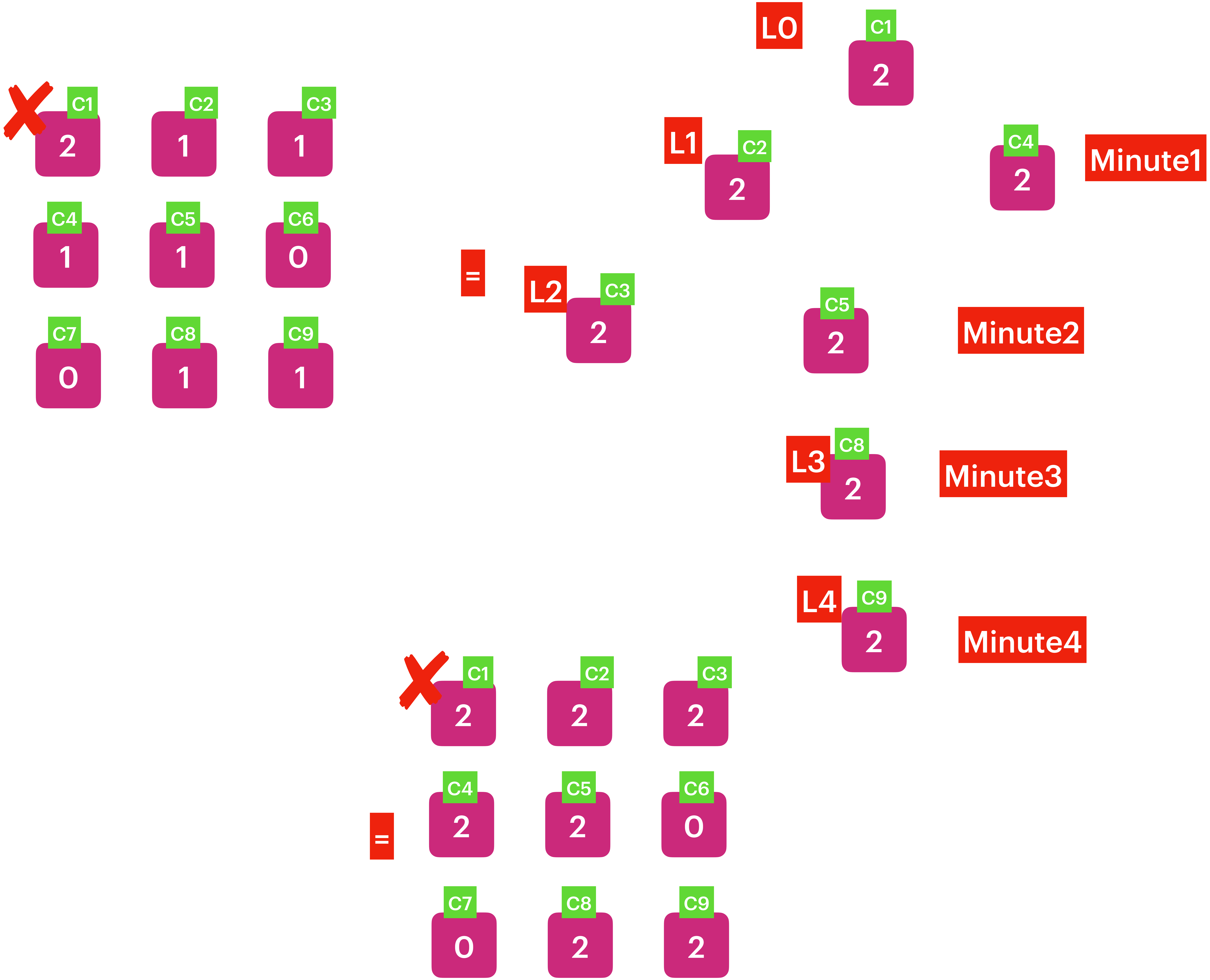
	C0	C1	C2
R0	2	2	2
R1	2	2	0
R2	0	1	1

Minute : 3

	C0	C1	C2
R0	2	2	2
R1	2	2	0
R2	0	2	1

Minute : 4

	C0	C1	C2
R0	2	2	2
R1	2	2	0
R2	0	2	2



C1 2	C2 1	C3 1
C4 1	C5 1	C6 0
C7 0	C8 1	C9 1

=

L0 C1 2

L1 C2 2

C4 2 Minute1

L2 C3 2

C5 2 Minute2

L3 C8 2 Minute3

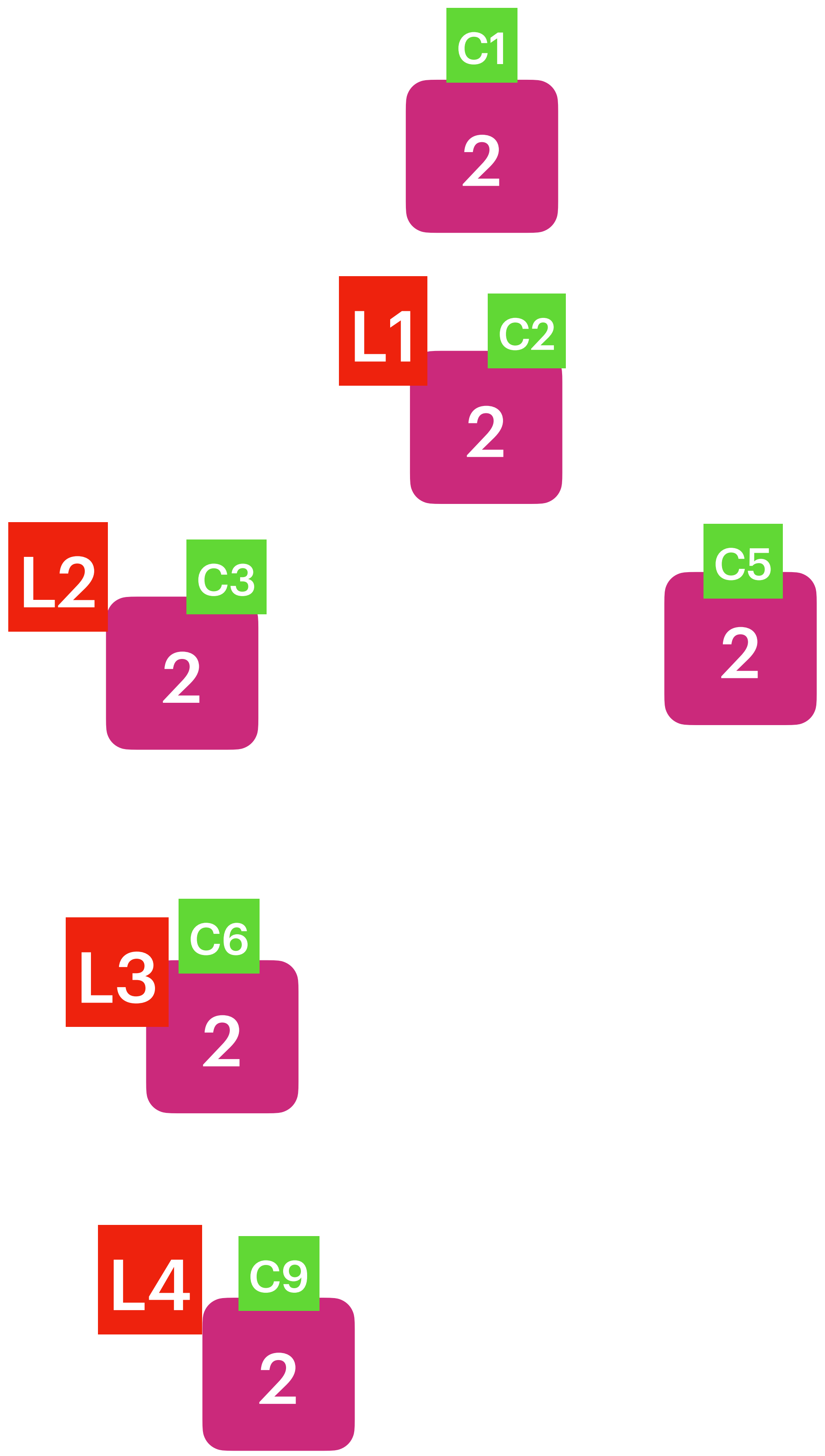
L4 C9 2 Minute4

=

C1 2	C2 2	C3 2
C4 2	C5 2	C6 0
C7 0	C8 2	C9 2

✗ C1 2	C2 1	C3 1
C4 0	C5 1	C6 1
C7 1	C8 0	C9 1

=



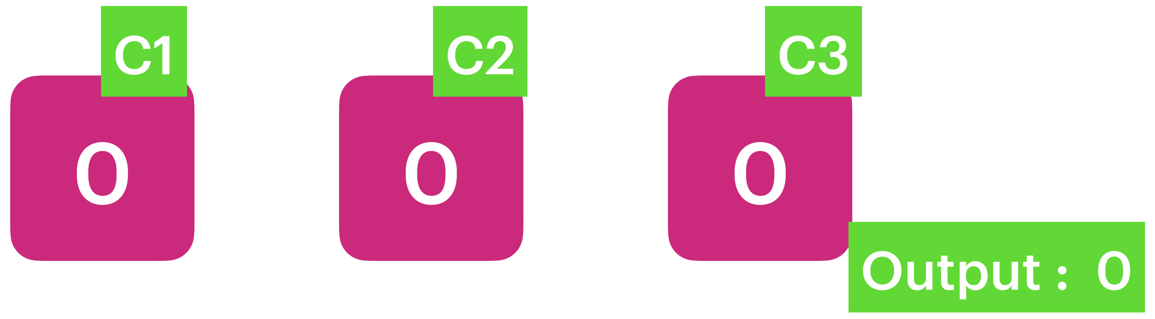
=

✗ C1 2	C2 2	C3 2
C4 0	C5 2	C6 2
C7 1	C8 0	C9 2

=

Output : -1

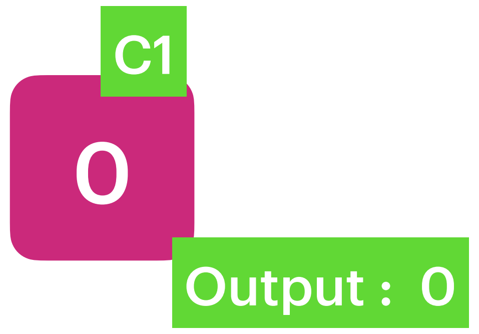
Cell7 is not rotated



C1 C2 C3

0 0 0

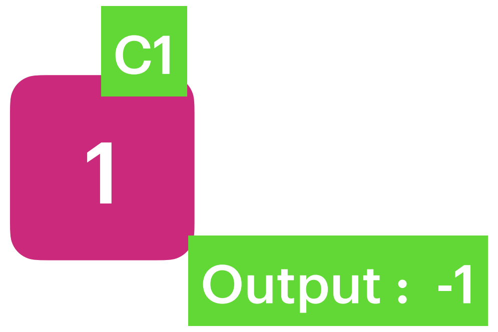
Output : 0



C1

0

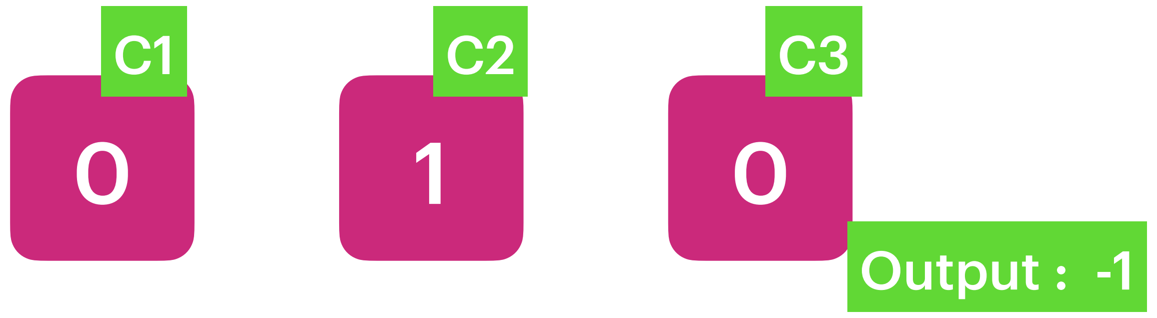
Output : 0



C1

1

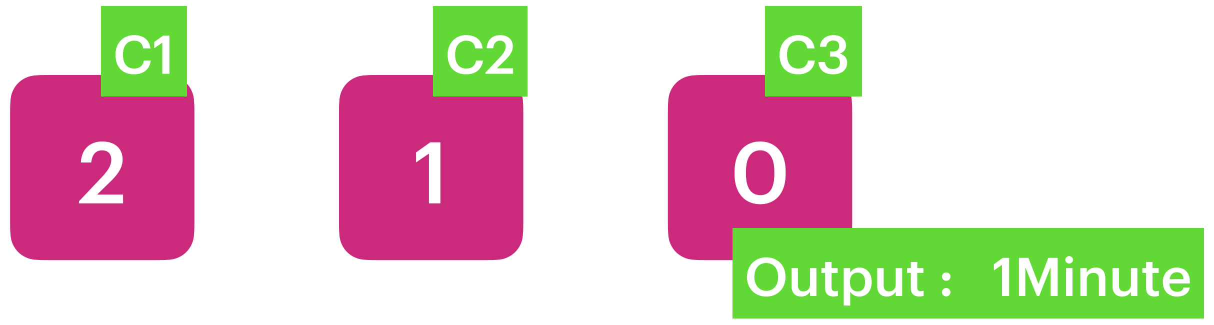
Output : -1



C1 C2 C3

0 1 0

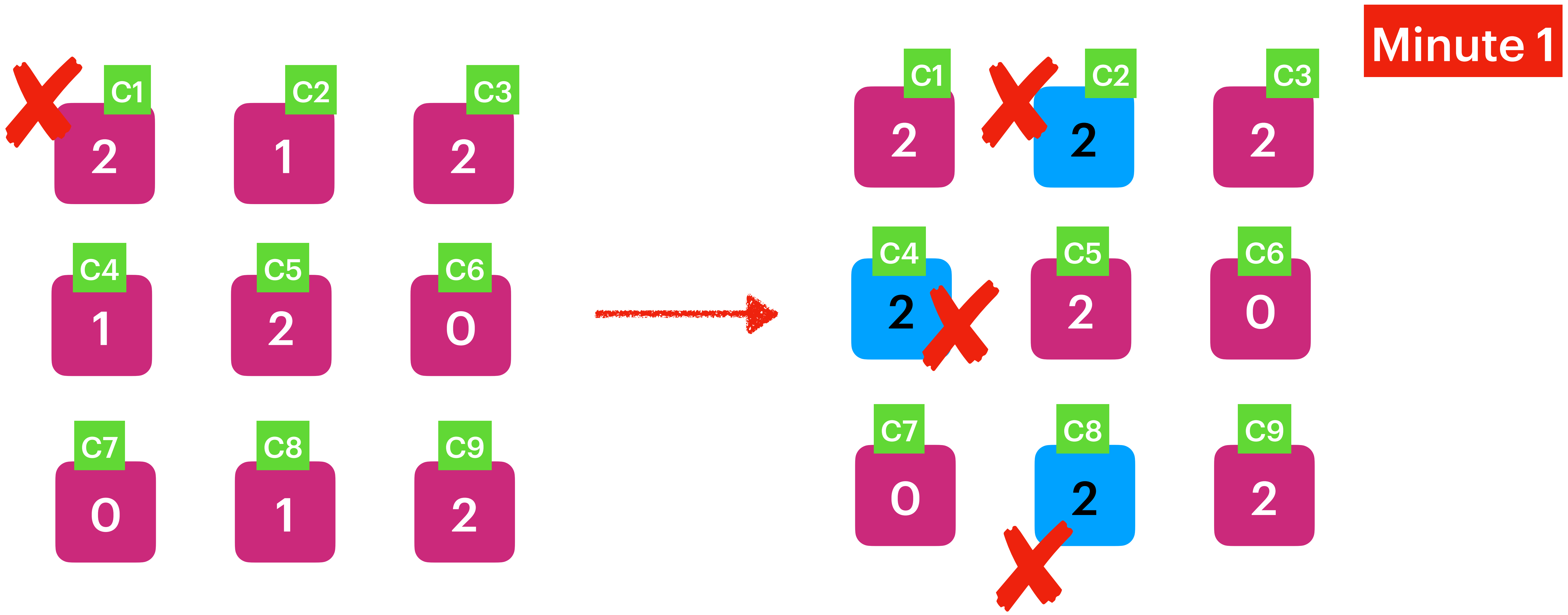
Output : -1

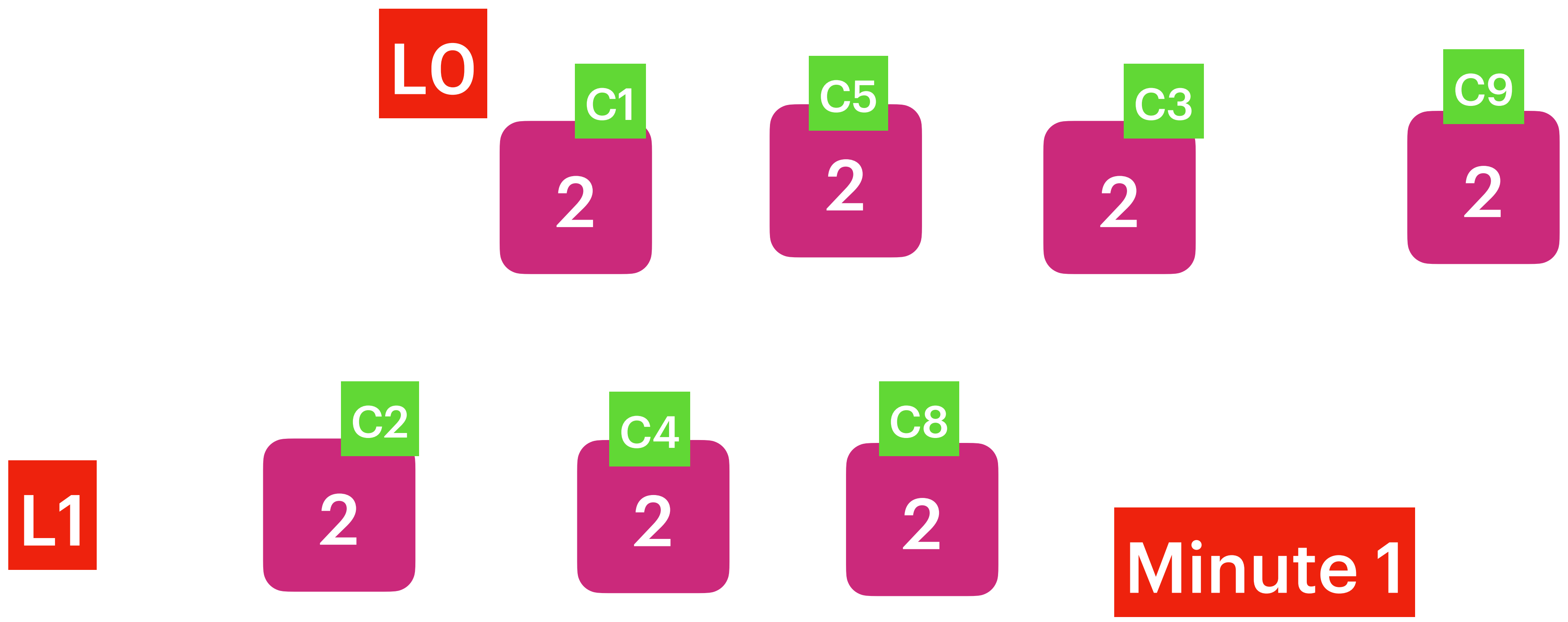


C1 C2 C3

2 1 0

Output : 1Minute


C1
2
C2
1
C3
2
C4
1
C5
2
C6
0
C7
0
C8
1
C9
2
C1
2
C2
2
C3
2
C4
2
C5
2
C6
0
C7
0
C8
2
C9
2
Minute 1


LO
C1
2
C5
2
C3
2
C9
2
L1
C2
2
C4
2
C8
2
Minute 1

Number of Islands

Given an m x n 2D binary grid grid which represents a map of '1's (land) and '0's (water), return the number of islands.

An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically.
You may assume all four edges of the grid are all surrounded by water.

```
Input: grid = [
 ["1","1","1","1","0"],
 ["1","1","0","1","0"],
 ["1","1","0","0","0"],
 ["0","0","0","0","0"]
]
Output: 1
```

```
Input: grid = [
 ["1","1","0","0","0"],
 ["1","1","0","0","0"],
 ["0","0","1","0","0"],
 ["0","0","0","1","1"]
]
Output: 3
```

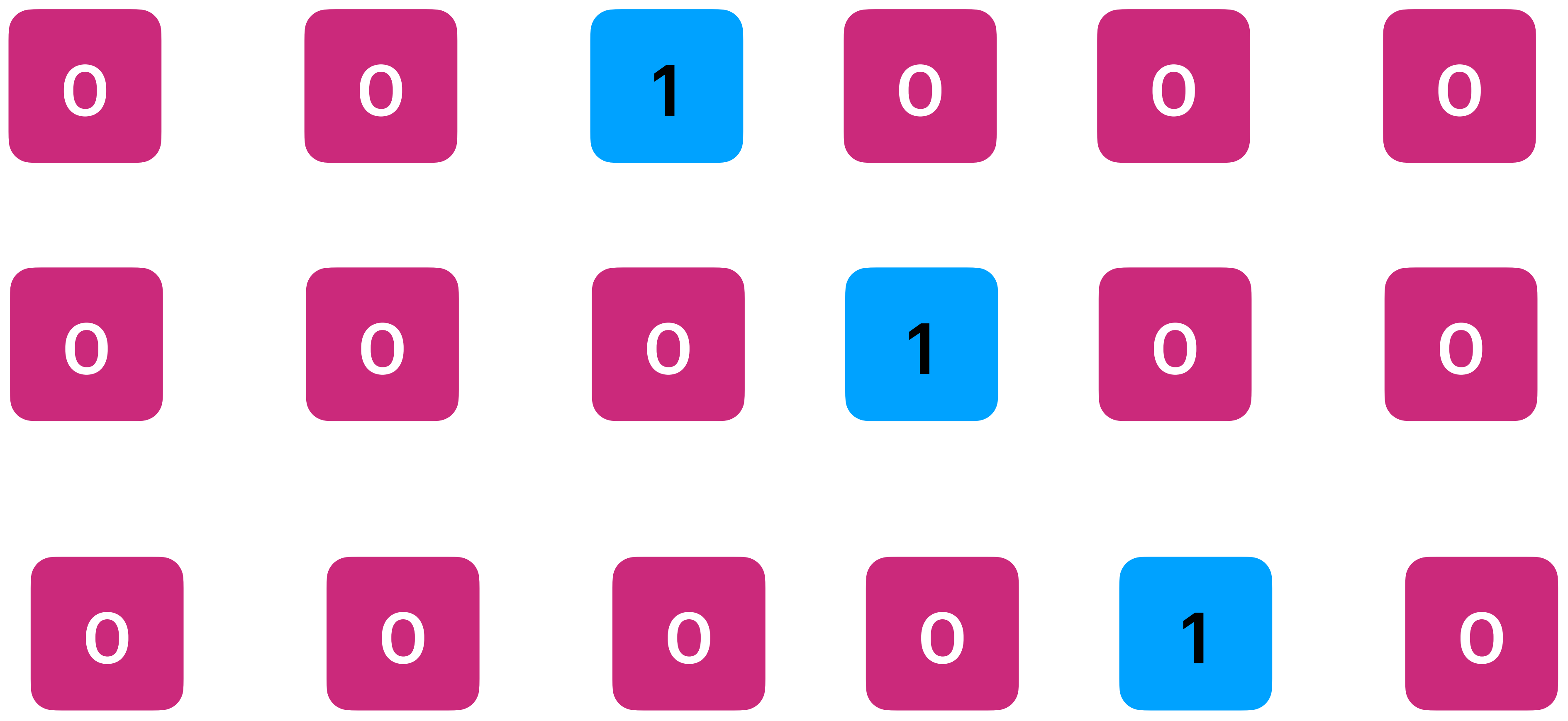
m == grid.length
n == grid[i].length
1 <= m, n <= 300
grid[i][j] is '0' or '1'.

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

islandsCount : 0

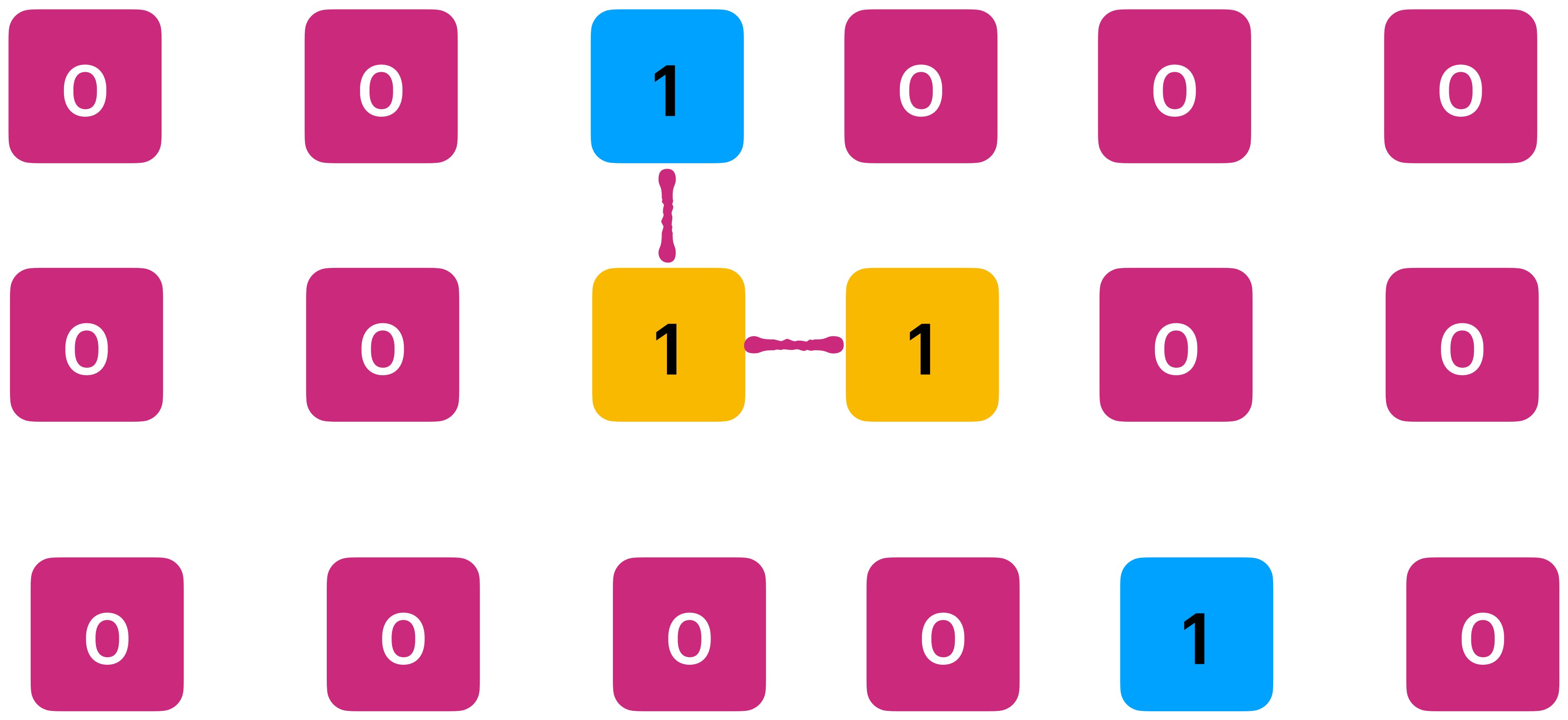
0	0	0	0	0	0
0	0	0	1	0	0
0	0	0	0	0	0

islandsCount : 1



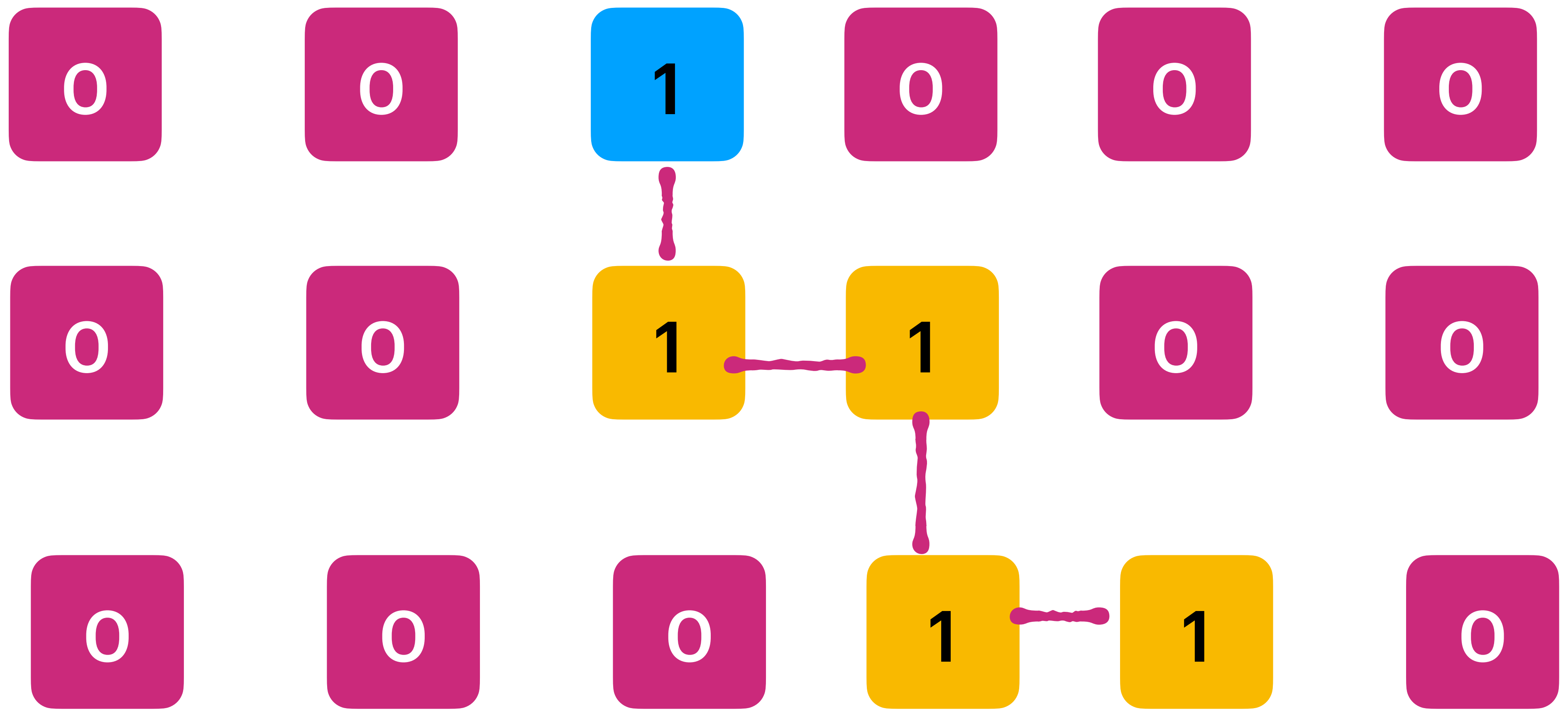
0	0	1	0	0	0
0	0	0	1	0	0
0	0	0	0	1	0

islandsCount : 3



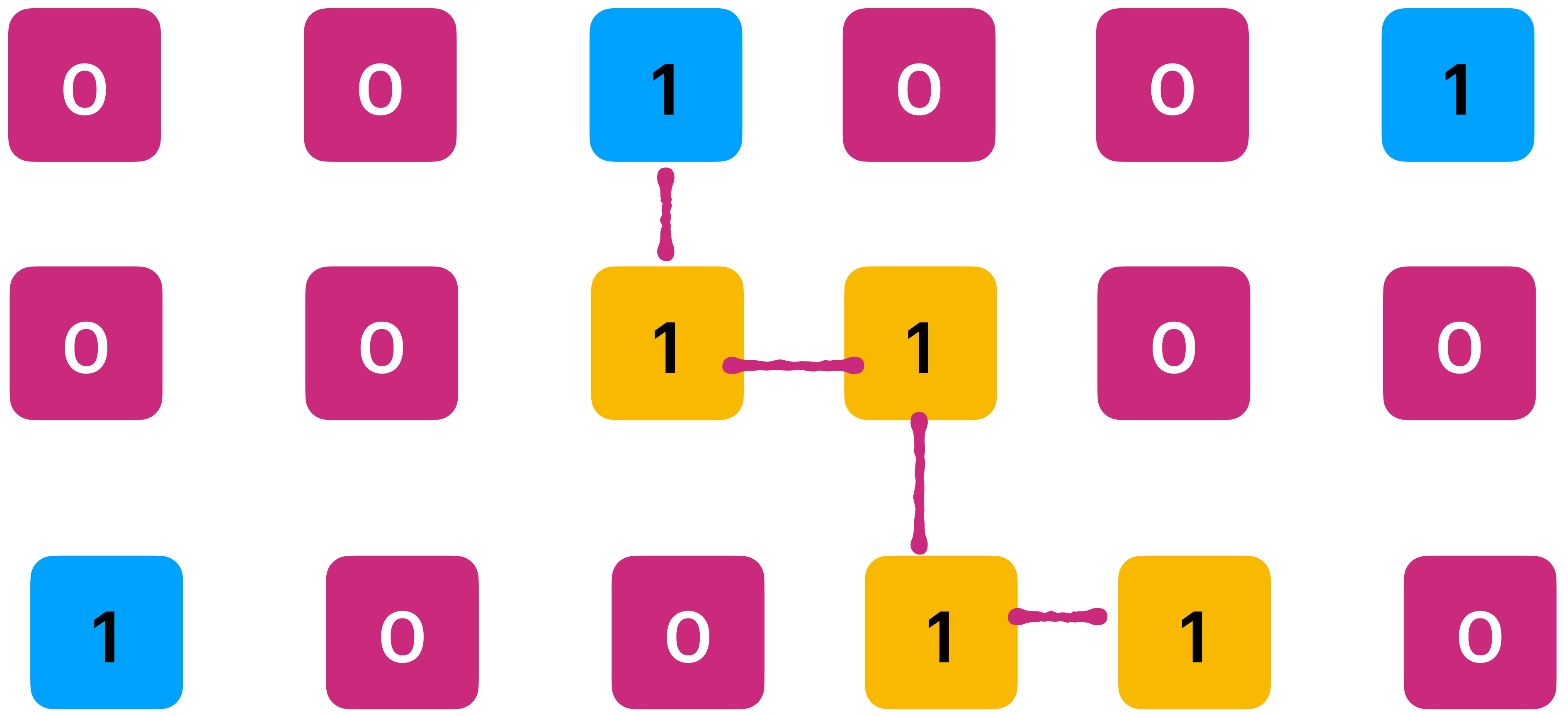
0	0	1	0	0	0
0	0	1	1	0	0
0	0	0	0	1	0

islandsCount : 2



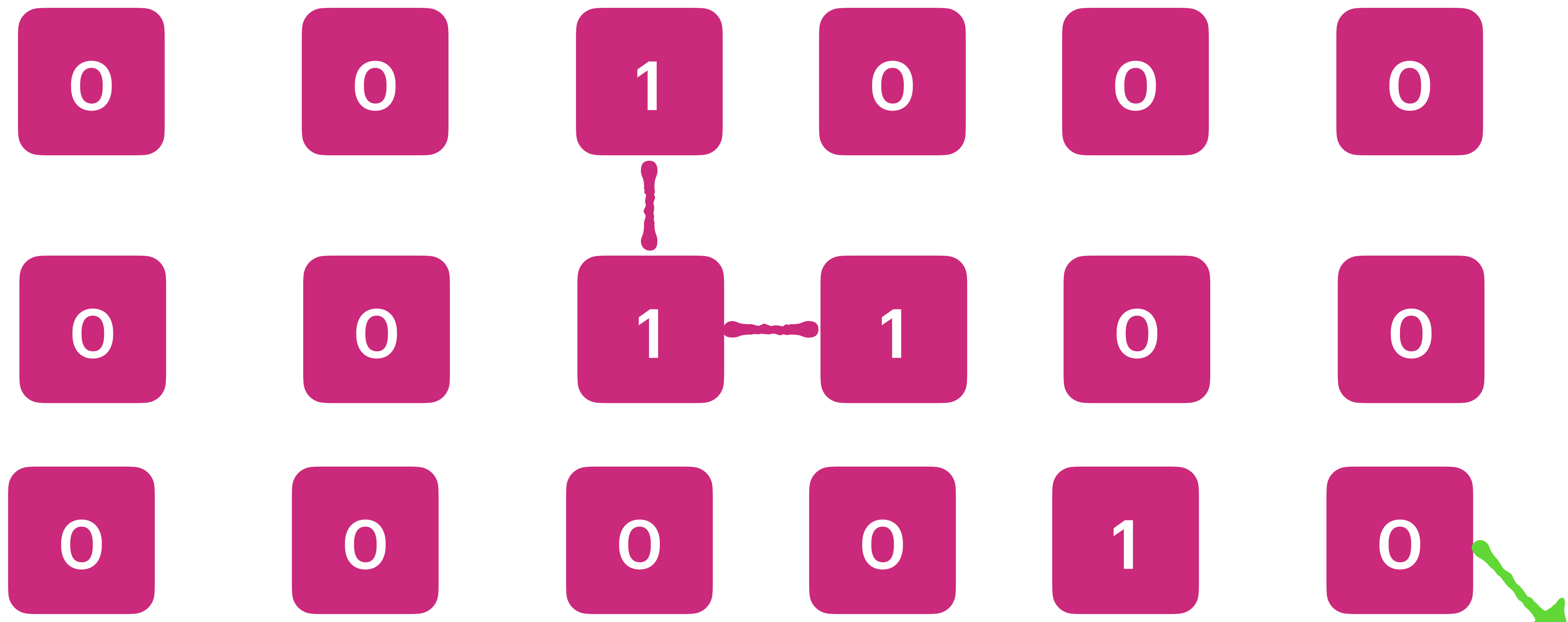
0	0	1	0	0	0
0	0	1	1	0	0
0	0	0	1	1	0

islandsCount : 1



0	0	1	0	0	1
0	0	1	1	0	0
1	0	0	1	1	0

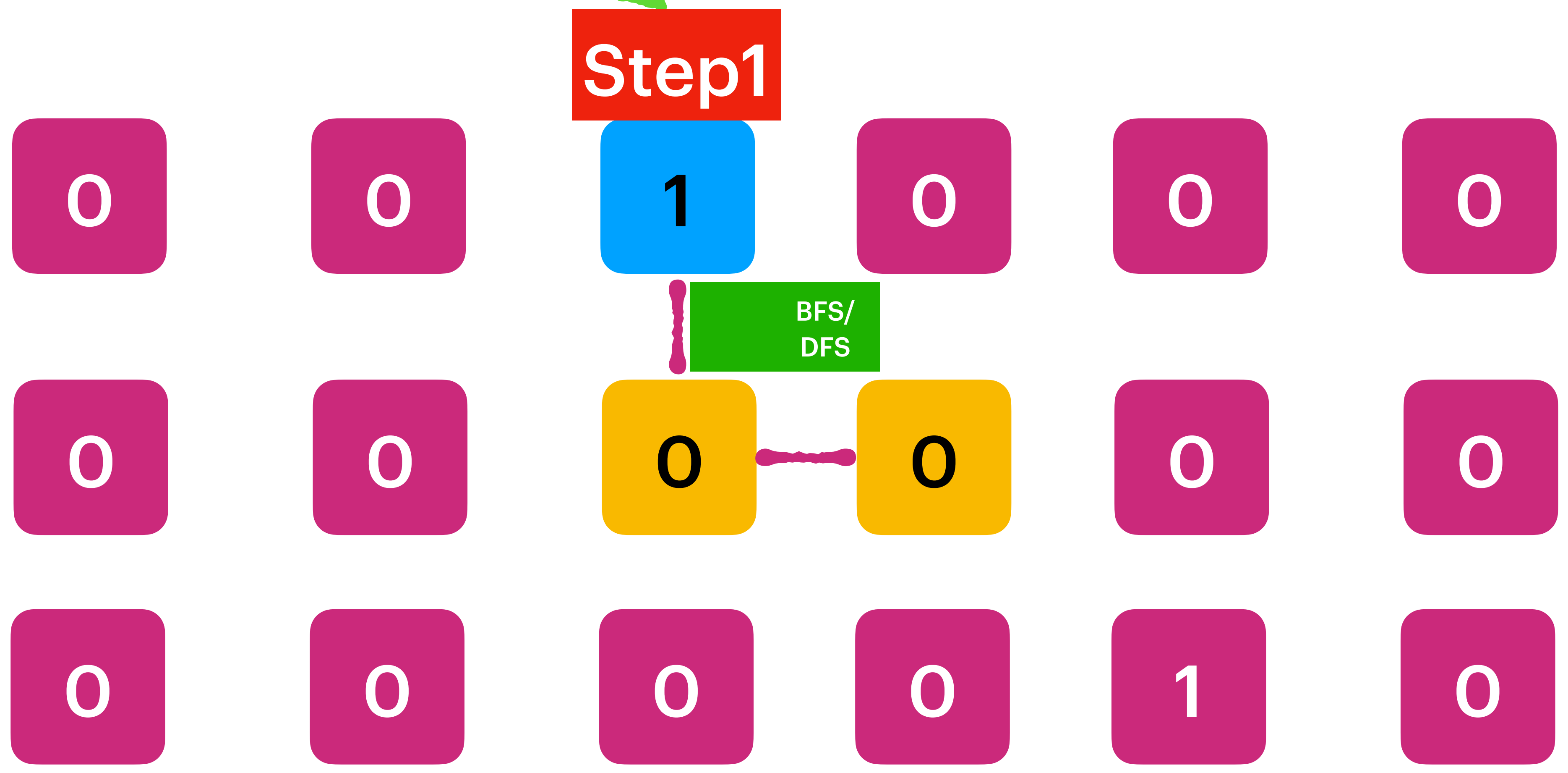
islandsCount : 3



0	0	1	0	0	0
0	0	1	1	0	0
0	0	0	0	1	0

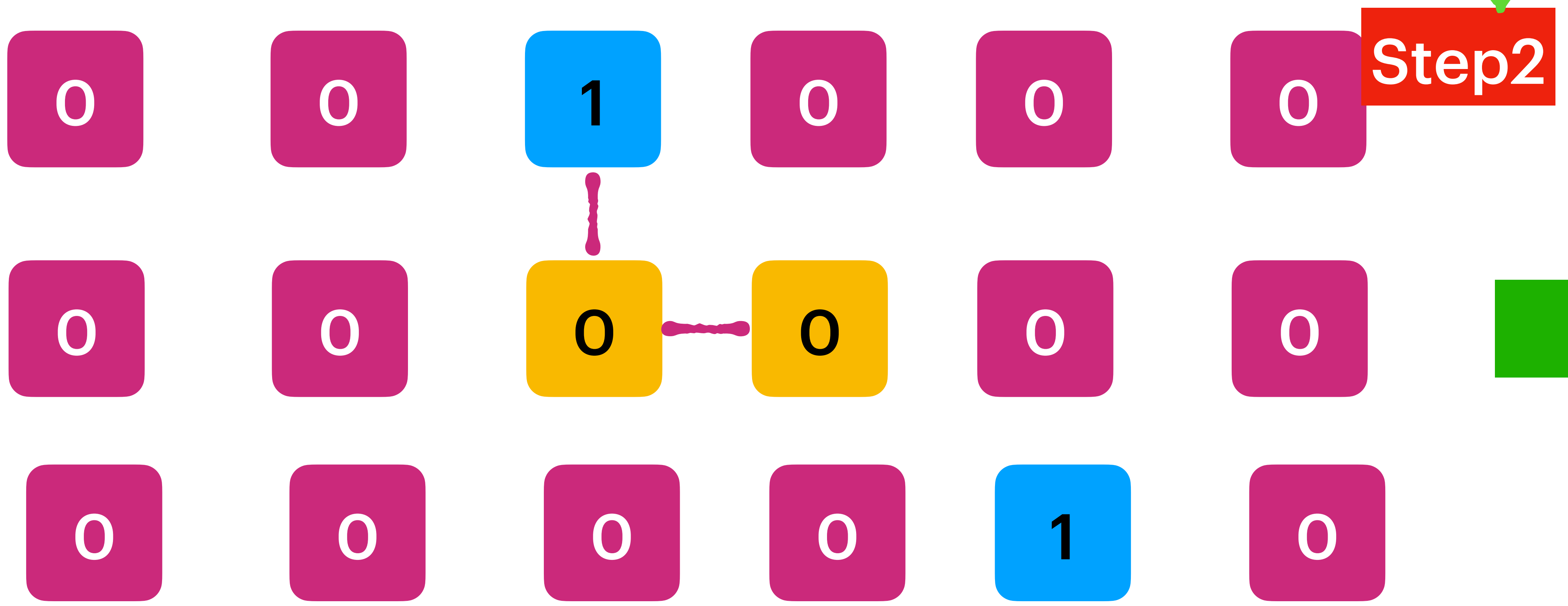
Total Islands : 2

```
for ( r : Row )
{
    for(r : Col)
    {
    if(cell[r][c] = bfs[r,c];
    islandCount+
    }
}
```



0	0	1	0	0	0
0	0	0	0	0	0
0	0	0	0	1	0

islandCount = 1



0	0	1	0	0	0
0	0	0	0	0	0
0	0	0	0	1	0

islandCount = 2

Shortest Path in Binary Matrix

Given an n x n binary matrix grid, return the length of the shortest clear path in the matrix.
If there is no clear path, return -1.
A clear path in a binary matrix is a path from the top-left cell (i.e., (0, 0)) to the bottom-right cell (i.e., (n - 1, n - 1)) such that:
All the visited cells of the path are 0.
All the adjacent cells of the path are 8-directionally connected (i.e., they are different and they share an edge or a corner).
The length of a clear path is the number of visited cells of this path.

Input: grid = [[0,1],[1,0]]
Output: 2

Input: grid = [[0,0,0],[1,1,0],[1,1,0]]
Output: 4

Input: grid = [[1,0,0],[1,1,0],[1,1,0]]
Output: -1

Constraints:
n == grid.length
n == grid[i].length
1 <= n <= 100
grid[i][j] is 0 or 1