# 1765. Map of Highest Peak

Medium | Topics | Companies | Hint

You are given an integer matrix `isWater` of size `m x n` that represents a map of **land** and **water** cells.

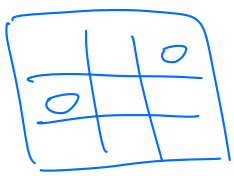
- If `isWater[i][j] == 0`, cell `(i, j)` is a **land** cell.
- If `isWater[i][j] == 1`, cell `(i, j)` is a **water** cell.

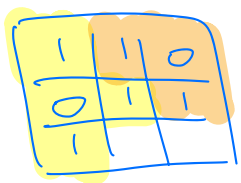
You must assign each cell a height in a way that follows these rules:

- The height of each cell must be non-negative.
- If the cell is a **water** cell, its height must be `0`.
- Any two adjacent cells must have an absolute height difference of **at most** `1`. A cell is adjacent to another cell if the former is directly north, east, south, or west of the latter (i.e., their sides are touching).

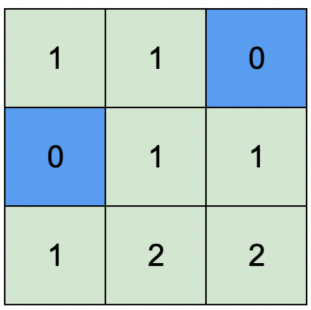
Find an assignment of heights such that the maximum height in the matrix is **maximized**.

Return *an integer matrix* `height` *of size* `m x n` *where* `height[i][j]` *is cell* `(i, j)`*'s height. If there are multiple solutions, return* ***any*** *of them.*





**Example 2:**




```
Input: isWater = [[0,0,1],[1,0,0],[0,0,0]]
Output: [[1,1,0],[0,1,1],[1,2,2]]
Explanation: A height of 2 is the maximum possible height of any assignment.
Any height assignment that has a maximum height of 2 while still meeting the rules will also be accepted.
```

Description | Accepted | Editorial | Solutions | Submissions

All Submissions

Accepted 59 / 59 testcases passed

AndrewC275 submitted at Jan 22, 2025 10:16

Editorial | Solution

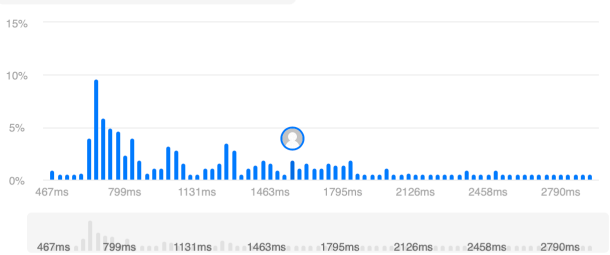
Runtime

1570 ms | Beats 29.17%

Analyze Complexity

Memory

221.70 MB | Beats 22.82%




Code | Python3

```python
class Solution:
    def highestPeak(self, isWater: List[List[int]]) -> List[List[int]]:
        m, n = len(isWater), len(isWater[0]) # row, col
        #BFS from water
        q = deque()
        visited = set()
```

Code

Python3 | Auto

```python
class Solution:
    def highestPeak(self, isWater: List[List[int]]) -> List[List[int]]:
        m, n = len(isWater), len(isWater[0]) # row, col
        #BFS from water
        q = deque()
        visited = set()
        res = [[-1] * n for _ in range(m)]

        for i in range(m):
            for j in range(n):
                if isWater[i][j] == 1:
                    q.append((i,j))
                    visited.add((i, j))
                    res[i][j] = 0

        while q:
            i, j = q.popleft()
            h = res[i][j]

            adj = [[i+1, j], [i-1, j], [i, j+1], [i, j-1]]
            for dx, dy in adj:
                if dx < 0 or dy < 0 or dx == m or dy == n or (dx, dy) in visited:
                    continue

                q.append((dx, dy))
                visited.add((dx,dy))
                res[dx][dy] =  h + 1


        return res
```