# 1765. Map of Highest Peak

***(same: 542. 01 Matrix)***

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:

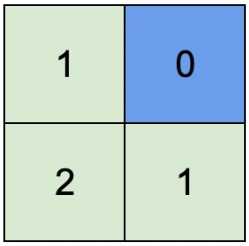
- The height of each cell must be non-negative.
- If the cell is a **water** cell, its height must be `0`.
- Any two adjacent cells must have an absolute height difference of **at most** `1`. A cell is adjacent to another cell if the former is directly north, east, south, or west of the latter (i.e.,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

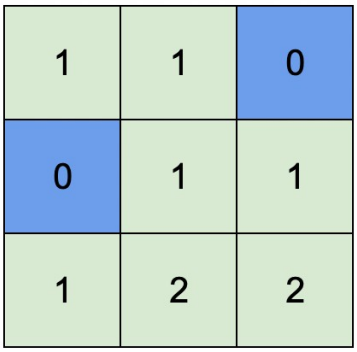
**Example 1:**

```
Input: isWater = [[0,1],[0,0]]
Output: [[1,0],[2,1]]
Explanation: The image shows the assigned heights of each cell.
The blue cell is the water cell, and the green cells are the land cells.
```




**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.
```



**Constraints:**

- `m == isWater.length`
- `n == isWater[i].length`
- `1 <= m, n <= 1000`
- `isWater[i][j]` is `0` or `1`.
- There is at least **one** water cell.

## Overview

We are given a 2D matrix `isWater` of dimensions `m x n`, which represents a map consisting of land and water cells. Specifically:

- If `isWater[i][j] = 0`, the cell `(i, j)` represents land.
- If `isWater[i][j] = 1`, the cell `(i, j)` represents water.

The goal is to assign a height to each cell such that the highest peak on the map (i.e., the greatest height of any cell) is as high as possible. This assignment must follow these rules:

1. The height of each cell must be non-negative.
2. The height of all water cells is fixed at 0. These cells have fixed heights and cannot be changed.
3. The height difference between two adjacent cells (cells that share a side) must not be greater than one. For example, if the height of cell `(2, 3)` is `4`, then the heights of its adjacent cells—`(1, 3)`, `(3, 3)`, `(2, 4)`, and `(2, 2)`—must be either `3`, `4` or `5`.

# 1765. Map of Highest Peak

```
/*
   Multi source BFS
   (nearest distance from a water cell to a land cell)
   Time compelxity: O(2mn)
   Space complexity: O(mn)
*/
typedef std::vector<int> vi;
typedef std::vector<vi> vvi;
typedef std::pair<int,int> ii;
class Solution {
   public:
      vvi highestPeak(vvi& isWater){
         vvi directions={{0,1},{0,-1},{-1,0},{1,0}};
         int m=isWater.size();
         int n=isWater[0].size();
         vvi ans(m,vi(n,-1));

         // Add all water cells to the queue and set their height to 0
         std::queue<ii> q;
         for(int row=0;row<m;++row){
            for(int col=0;col<n;++col){
               if(isWater[row][col]){
                  q.push({row,col});
                  ans[row][col]=0;
               }
            }
         }

         while(!q.empty()){
            // For each cell...
            auto [cur_row,cur_col]=q.front();
            q.pop();
            // ...get its neighbors(right,left,up,down)
            // For each neighbor...
            for(auto& dir: directions){
               int row=cur_row+dir[0];
               int col=cur_col+dir[1];

               if(row<0 || col<0 || row>m-1 || col>n-1) continue;
               if(ans[row][col]!=-1) continue; // Skip  it, If cell is visited

               // ...compute the distance to the nearest water cell
               ans[row][col]=ans[cur_row][cur_col]+1;

               q.push({row,col});
            }
         }
         return ans;
      }
};
```