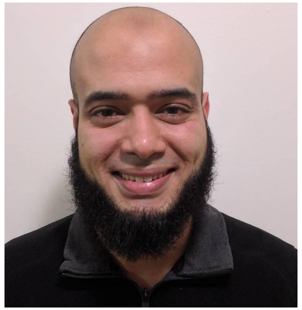
# *Algorithms*

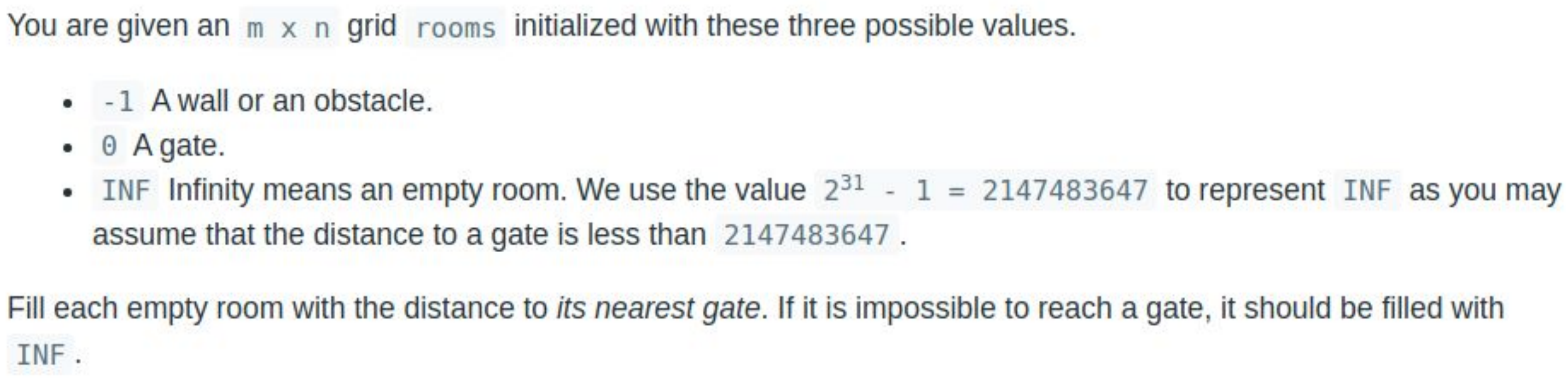
# BFS Homework 3

**Mostafa S. Ibrahim**
*Teaching, Training and Coaching for more than a decade!*

*Artificial Intelligence & Computer Vision Researcher*
*PhD* from Simon Fraser University - Canada
*Bachelor / Msc* from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)



## Problem #1: LeetCode 286 - Walls and Gates



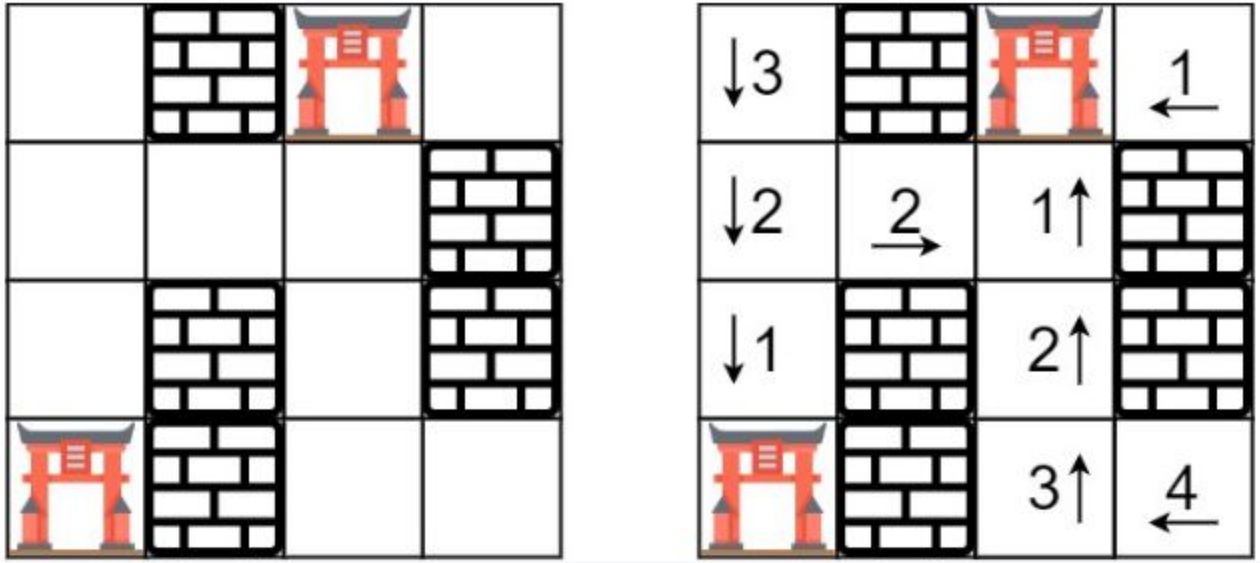
You are given an `m x n` grid `rooms` initialized with these three possible values.

- `-1` A wall or an obstacle.
- `0` A gate.
- `INF` Infinity means an empty room. We use the value `2^31 - 1 = 2147483647` to represent `INF` as you may assume that the distance to a gate is less than `2147483647`.

Fill each empty room with the distance to *its nearest gate*. If it is impossible to reach a gate, it should be filled with `INF`.

- C++: void wallsAndGates(vector<vector<int>>& rooms)
- Java: public void wallsAndGates(int[][] rooms)
- Python: def wallsAndGates(self, rooms: List[List[int]]) -> None
- Javascript: var wallsAndGates = function(rooms)

**Example 1:**



```
Input: rooms = [[2147483647,-1,0,2147483647],[2147483647,2147483647,2147483647,-1],
[2147483647,-1,2147483647,-1],[0,-1,2147483647,2147483647]]
Output: [[3,-1,0,1],[2,2,1,-1],[1,-1,2,-1],[0,-1,3,4]]
```

**Example 2:**

```
Input: rooms = [[-1]]
Output: [[-1]]
```

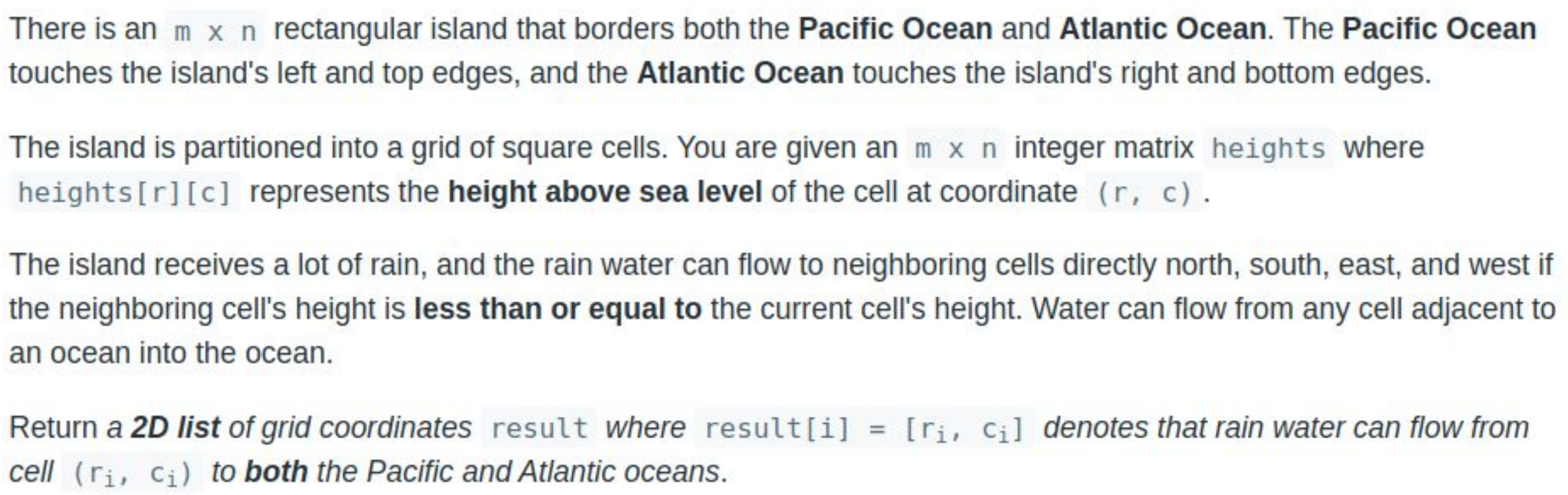
**Example 3:**

```
Input: rooms = [[2147483647]]
Output: [[2147483647]]
```

**Example 4:**

```
Input: rooms = [[0]]
Output: [[0]]
```

# Problem #2: LeetCode 417 - Pacific Atlantic Water Flow



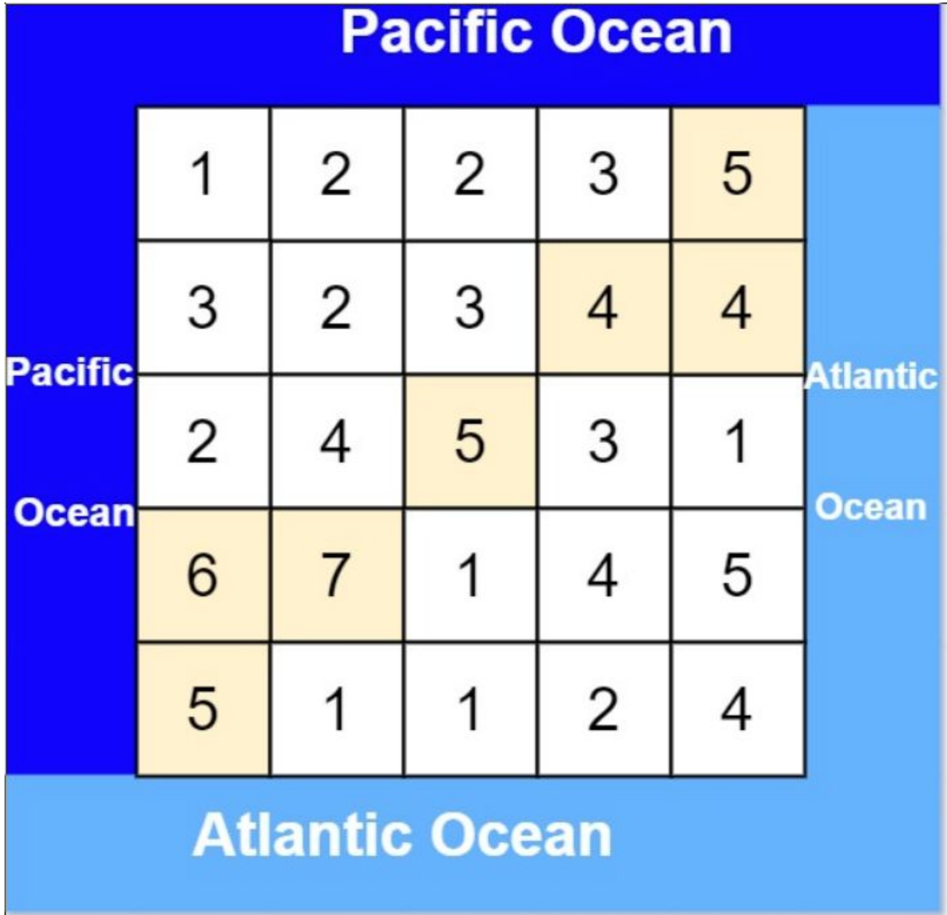
There is an `m x n` rectangular island that borders both the **Pacific Ocean** and **Atlantic Ocean**. The **Pacific Ocean** touches the island's left and top edges, and the **Atlantic Ocean** touches the island's right and bottom edges.

The island is partitioned into a grid of square cells. You are given an `m x n` integer matrix `heights` where `heights[r][c]` represents the **height above sea level** of the cell at coordinate `(r, c)`.

The island receives a lot of rain, and the rain water can flow to neighboring cells directly north, south, east, and west if the neighboring cell's height is **less than or equal to** the current cell's height. Water can flow from any cell adjacent to an ocean into the ocean.

Return *a **2D list** of grid coordinates* `result` *where* `result[i]` $= [r_i, c_i]$ *denotes that rain water can flow from cell* $(r_i, c_i)$ *to **both** the Pacific and Atlantic oceans.*

- C++: vector<vector<int>> pacificAtlantic(vector<vector<int>>& heights)
- Java: public List<List<Integer>> pacificAtlantic(int[][] heights)
- Python: def pacificAtlantic(self, heights: List[List[int]]) -> List[List[int]]
- Javascript: var pacificAtlantic = function(heights)

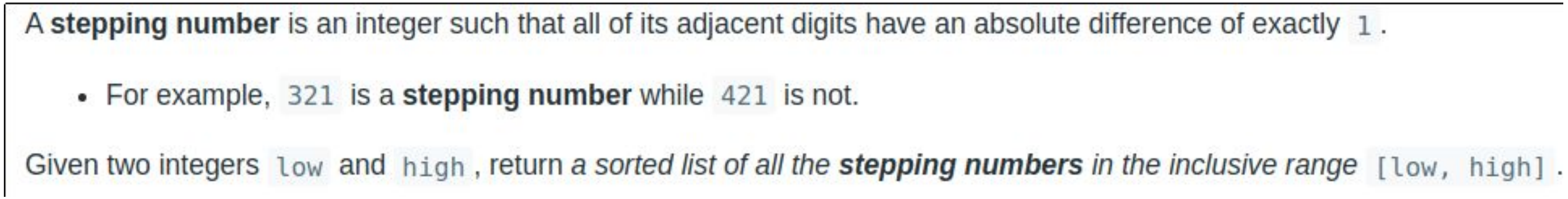


```
Input: heights = [[1,2,2,3,5],[3,2,3,4,4],[2,4,5,3,1],[6,7,1,4,5],[5,1,1,2,4]]
Output: [[0,4],[1,3],[1,4],[2,2],[3,0],[3,1],[4,0]]
```

**Example 2:**

```
Input: heights = [[2,1],[1,2]]
Output: [[0,0],[0,1],[1,0],[1,1]]
```

# Problem #3: LeetCode 1215 - Stepping Numbers



A **stepping number** is an integer such that all of its adjacent digits have an absolute difference of exactly `1`.

- For example, `321` is a **stepping number** while `421` is not.

Given two integers `low` and `high`, return *a sorted list of all the **stepping numbers** in the inclusive range* `[low, high]`.

- C++: vector<int> countSteppingNumbers(LL low, LL high)
- Java: public List<Integer> countSteppingNumbers(int low, int high)
- Python: def countSteppingNumbers(self, low: int, high: int) -> List[int]
- Javascript: var countSteppingNumbers = function(low, high)
- $0 <= low <= high <= 2 * 10^9$

**Example 1:**

```
Input: low = 0, high = 21
Output: [0,1,2,3,4,5,6,7,8,9,10,12,21]
```

**Example 2:**

```
Input: low = 10, high = 15
Output: [10,12]
```

*“Acquire knowledge and impart it to the people.”*

*“Seek knowledge from the Cradle to the Grave.”*