286. Walls and Gates

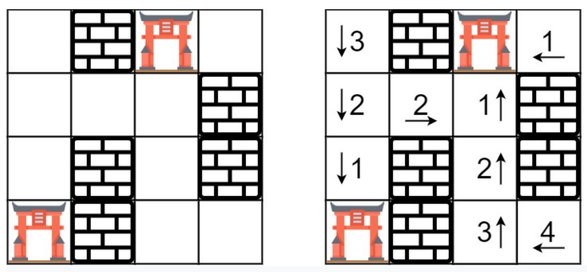
Medium 1945 27 Add to List Share

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`.

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]]
```

Constraints:

- `m == rooms.length`
- `n == rooms[i].length`
- `1 <= m, n <= 250`
- `rooms[i][j]` is `-1`, `0`, or $2^{31} - 1$.

Accepted 184,217 | Submissions 317,693

Seen this question in a real interview before? Yes No

Companies

0 ~ 6 months 6 months ~ 1 year 1 year ~ 2 years

Facebook | 10 Amazon | 4 DoorDash | 4 Uber | 2 Spotify | 2

Related Topics

Array Breadth-First Search Matrix

Similar Questions

Surrounded Regions	Medium
Number of Islands	Medium
Shortest Distance from All Buildings	Hard
Robot Room Cleaner	Hard
Rotting Oranges	Medium

Java Autocomplete

```java
class Solution {
    private static final int EMPTY = Integer.MAX_VALUE;
    private static final int GATE = 0;
    private static final List<int[]> DIRECTIONS = Arrays.asList(
        new int[] { 1,  0},
        new int[] {-1,  0},
        new int[] { 0,  1},
        new int[] { 0, -1}
    );

    public void wallsAndGates(int[][] rooms) {
        int m = rooms.length;
    if (m == 0) return;
    int n = rooms[0].length;
    Queue<int[]> q = new LinkedList<>();
    for (int row = 0; row < m; row++) {
        for (int col = 0; col < n; col++) {
            if (rooms[row][col] == GATE) {
                q.add(new int[] { row, col });
            }
        }
    }
    while (!q.isEmpty()) {
        int[] point = q.poll();
        int row = point[0];
        int col = point[1];
        for (int[] direction : DIRECTIONS) {
            int r = row + direction[0];
            int c = col + direction[1];
            if (r < 0 || c < 0 || r >= m || c >= n || rooms[r][c] != EMPTY) {
                continue;
            }
            rooms[r][c] = rooms[row][col] + 1;
            q.add(new int[] { r, c });
        }
    }
    }
}
```