

# Problem 286: Walls and Gates

## Problem Information

**Difficulty:** Medium

**Acceptance Rate:** 0.00%

**Paid Only:** No

## Problem Description

You are given an

$m \times n$

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

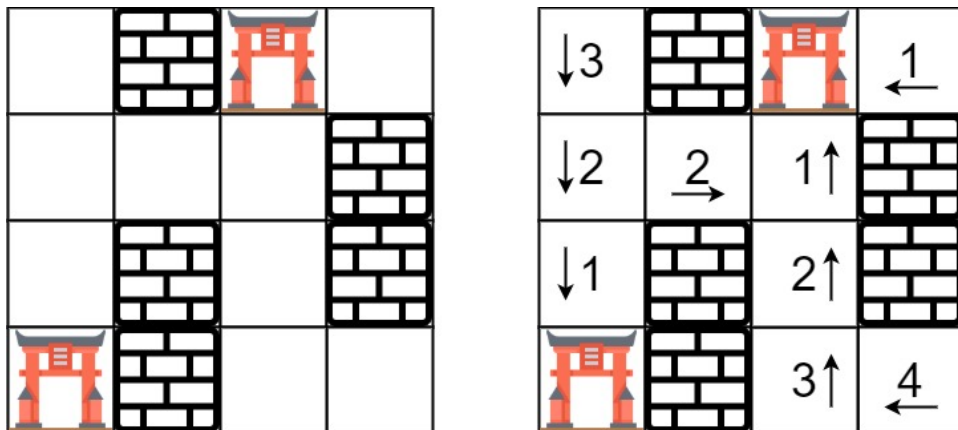
its nearest gate

. 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]]
```

Constraints:

```
m == rooms.length
```

```
n == rooms[i].length
```

```
1 <= m, n <= 250
```

```
rooms[i][j]
```

is

-1

,

0

, or

2

31

- 1

## Code Snippets

### C++:

```
class Solution {
public:
    void wallsAndGates(vector<vector<int>>& rooms) {

    }
};
```

### Java:

```
class Solution {
    public void wallsAndGates(int[][] rooms) {

    }
}
```

### Python3:

```
class Solution:
    def wallsAndGates(self, rooms: List[List[int]]) -> None:
        """
        Do not return anything, modify rooms in-place instead.
        """
```

### Python:

```
class Solution(object):
    def wallsAndGates(self, rooms):
        """
        :type rooms: List[List[int]]
        :rtype: None Do not return anything, modify rooms in-place instead.
        """
```

### JavaScript:

```

/**
 * @param {number[][]} rooms
 * @return {void} Do not return anything, modify rooms in-place instead.
 */
var wallsAndGates = function(rooms) {

};

```

### TypeScript:

```

/**
 Do not return anything, modify rooms in-place instead.
 */
function wallsAndGates(rooms: number[][]): void {

};

```

### C#:

```

public class Solution {
    public void WallsAndGates(int[][] rooms) {

    }
}

```

### C:

```

void wallsAndGates(int** rooms, int roomsSize, int* roomsColSize) {

}

```

### Go:

```

func wallsAndGates(rooms [][]int) {

}

```

### Kotlin:

```

class Solution {
    fun wallsAndGates(rooms: Array<IntArray>): Unit {

    }
}

```

```
}
```

### Swift:

```
class Solution {  
    func wallsAndGates(_ rooms: inout [[Int]]) {  
  
    }  
}
```

### Rust:

```
impl Solution {  
    pub fn walls_and_gates(rooms: &mut Vec<Vec<i32>> ) {  
  
    }  
}
```

### Ruby:

```
# @param {Integer[][]} rooms  
# @return {Void} Do not return anything, modify rooms in-place instead.  
def walls_and_gates(rooms)  
  
end
```

### PHP:

```
class Solution {  
  
    /**  
     * @param Integer[][] $rooms  
     * @return NULL  
     */  
    function wallsAndGates(&$rooms) {  
  
    }  
}
```

### Dart:

```

class Solution {
void wallsAndGates(List<List<int>> rooms) {

}

}

```

### Scala:

```

object Solution {
def wallsAndGates(rooms: Array[Array[Int]]): Unit = {

}

}

```

## Solutions

### C++ Solution:

```

/*
 * Problem: Walls and Gates
 * Difficulty: Medium
 * Tags: array, tree, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(h) for recursion stack where h is height
 */

class Solution {
public:
void wallsAndGates(vector<vector<int>>& rooms) {

}

};

```

### Java Solution:

```

/**
 * Problem: Walls and Gates
 * Difficulty: Medium
 * Tags: array, tree, search

```

```

*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public void wallsAndGates(int[][] rooms) {

}
}

```

### Python3 Solution:

```

"""
Problem: Walls and Gates
Difficulty: Medium
Tags: array, tree, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
def wallsAndGates(self, rooms: List[List[int]]) -> None:
# TODO: Implement optimized solution
pass

```

### Python Solution:

```

class Solution(object):
def wallsAndGates(self, rooms):
"""
:type rooms: List[List[int]]
:rtype: None Do not return anything, modify rooms in-place instead.
"""

```

### JavaScript Solution:

```

/**
 * Problem: Walls and Gates
 * Difficulty: Medium
 * Tags: array, tree, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(h) for recursion stack where h is height
 */

/**
 * @param {number[][]} rooms
 * @return {void} Do not return anything, modify rooms in-place instead.
 */
var wallsAndGates = function(rooms) {

};

```

### TypeScript Solution:

```

/**
 * Problem: Walls and Gates
 * Difficulty: Medium
 * Tags: array, tree, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(h) for recursion stack where h is height
 */

/**
Do not return anything, modify rooms in-place instead.
 */
function wallsAndGates(rooms: number[][]): void {

};

```

### C# Solution:

```

/*
 * Problem: Walls and Gates
 * Difficulty: Medium

```

```

* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public void WallsAndGates(int[][] rooms) {

}
}

```

### C Solution:

```

/*
* Problem: Walls and Gates
* Difficulty: Medium
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

void wallsAndGates(int** rooms, int roomsSize, int* roomsColSize) {

}

```

### Go Solution:

```

// Problem: Walls and Gates
// Difficulty: Medium
// Tags: array, tree, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func wallsAndGates(rooms [][]int) {

```

```
}
```

### Kotlin Solution:

```
class Solution {  
    fun wallsAndGates(rooms: Array<IntArray>): Unit {  
  
    }  
}
```

### Swift Solution:

```
class Solution {  
    func wallsAndGates(_ rooms: inout [[Int]]) {  
  
    }  
}
```

### Rust Solution:

```
// Problem: Walls and Gates  
// Difficulty: Medium  
// Tags: array, tree, search  
//  
// Approach: Use two pointers or sliding window technique  
// Time Complexity: O(n) or O(n log n)  
// Space Complexity: O(h) for recursion stack where h is height  
  
impl Solution {  
    pub fn walls_and_gates(rooms: &mut Vec<Vec<i32>>) {  
  
    }  
}
```

### Ruby Solution:

```
# @param {Integer[][]} rooms  
# @return {Void} Do not return anything, modify rooms in-place instead.  
def walls_and_gates(rooms)  
  
end
```

### PHP Solution:

```
class Solution {  
  
    /**  
     * @param Integer[][] $rooms  
     * @return NULL  
     */  
    function wallsAndGates(&$rooms) {  
  
    }  
}
```

### Dart Solution:

```
class Solution {  
    void wallsAndGates(List<List<int>> rooms) {  
  
    }  
}
```

### Scala Solution:

```
object Solution {  
    def wallsAndGates(rooms: Array[Array[Int]]): Unit = {  
  
    }  
}
```