Problem 994: Rotting Oranges

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

You are given an

m x n

grid

where each cell can have one of three values:

0

representing an empty cell,

1

representing a fresh orange, or

2

representing a rotten orange.

Every minute, any fresh orange that is

4-directionally adjacent

to a rotten orange becomes rotten.

Return

the minimum number of minutes that must elapse until no cell has a fresh orange

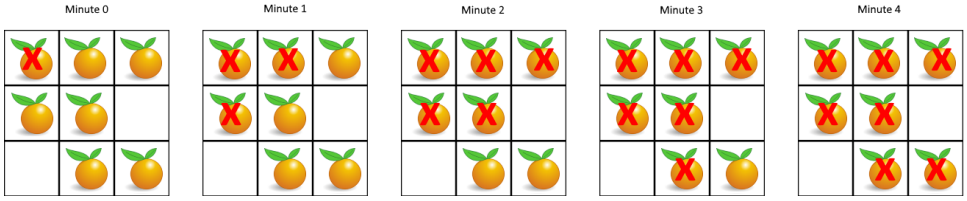
. If

this is impossible, return

-1

.

Example 1:



Input:

grid = [[2,1,1],[1,1,0],[0,1,1]]

Output:

4

Example 2:

Input:

grid = [[2,1,1],[0,1,1],[1,0,1]]

Output:

-1

Explanation:

The orange in the bottom left corner (row 2, column 0) is never rotten, because rotting only happens 4-directionally.

Example 3:

Input:

grid = [[0,2]]

Output:

0

Explanation:

Since there are already no fresh oranges at minute 0, the answer is just 0.

Constraints:

m == grid.length

n == grid[i].length

1 <= m, n <= 10

grid[i][j]

is

0

,

1

, or

2

.

Code Snippets

C++:

```
class Solution {
public:
int orangesRotting(vector<vector<int>>& grid) {

}
};
```

Java:

```
class Solution {
public int orangesRotting(int[][] grid) {

}
}
```

Python3:

```
class Solution:
def orangesRotting(self, grid: List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def orangesRotting(self, grid):
"""
:type grid: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} grid
* @return {number}
*/
```

```
var orangesRotting = function(grid) {

};
```

TypeScript:

```
function orangesRotting(grid: number[][]): number {

};
```

C#:

```
public class Solution {
public int OrangesRotting(int[][] grid) {

}
}
```

C:

```
int orangesRotting(int** grid, int gridSize, int* gridColSize) {

}
```

Go:

```
func orangesRotting(grid [][]int) int {

}
```

Kotlin:

```
class Solution {
fun orangesRotting(grid: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func orangesRotting(_ grid: [[Int]]) -> Int {
```

```
}
}
```

Rust:

```
impl Solution {
pub fn oranges_rotting(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} grid
# @return {Integer}
def oranges_rotting(grid)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $grid
* @return Integer
*/
function orangesRotting($grid) {

}
}
```

Dart:

```
class Solution {
int orangesRotting(List<List<int>> grid) {

}
}
```

Scala:

```
object Solution {
def orangesRotting(grid: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec oranges_rotting(grid :: [[integer]]) :: integer
def oranges_rotting(grid) do

end
end
```

Erlang:

```
-spec oranges_rotting(Grid :: [[integer()]]) -> integer().
oranges_rotting(Grid) ->
.
```

Racket:

```
(define/contract (oranges-rotting grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Rotting Oranges
* Difficulty: Medium
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
class Solution {
public:
int orangesRotting(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Rotting Oranges
* Difficulty: Medium
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int orangesRotting(int[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Rotting Oranges
Difficulty: Medium
Tags: array, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def orangesRotting(self, grid: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def orangesRotting(self, grid):
"""
:type grid: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Rotting Oranges
* Difficulty: Medium
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number[][]} grid
* @return {number}
*/
var orangesRotting = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Rotting Oranges
* Difficulty: Medium
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function orangesRotting(grid: number[][]): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Rotting Oranges
* Difficulty: Medium
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int OrangesRotting(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Rotting Oranges
* Difficulty: Medium
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int orangesRotting(int** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```
// Problem: Rotting Oranges
// Difficulty: Medium
```

```
// Tags: array, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func orangesRotting(grid [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun orangesRotting(grid: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func orangesRotting(_ grid: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Rotting Oranges
// Difficulty: Medium
// Tags: array, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn oranges_rotting(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} grid
# @return {Integer}
def oranges_rotting(grid)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $grid
* @return Integer
*/
function orangesRotting($grid) {

}
}
```

Dart Solution:

```
class Solution {
int orangesRotting(List<List<int>> grid) {

}
}
```

Scala Solution:

```
object Solution {
def orangesRotting(grid: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec oranges_rotting(grid :: [[integer]]) :: integer
def oranges_rotting(grid) do
```

```
end
end
```

Erlang Solution:

```
-spec oranges_rotting(Grid :: [[integer()]]) -> integer().
oranges_rotting(Grid) ->
.
```

Racket Solution:

```
(define/contract (oranges-rotting grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```