Problem 2258: Escape the Spreading Fire

Problem Information

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

Problem Description

You are given a

0-indexed

2D integer array

grid

of size

m x n

which represents a field. Each cell has one of three values:

0

represents grass,

1

represents fire,

2

represents a wall that you and fire cannot pass through.

You are situated in the top-left cell,

(0, 0)

, and you want to travel to the safehouse at the bottom-right cell,

(m - 1, n - 1)

. Every minute, you may move to an

adjacent

grass cell.

After

your move, every fire cell will spread to all

adjacent

cells that are not walls.

Return

the

maximum

number of minutes that you can stay in your initial position before moving while still safely reaching the safehouse

. If this is impossible, return

-1

. If you can

always

reach the safehouse regardless of the minutes stayed, return

10

9

.

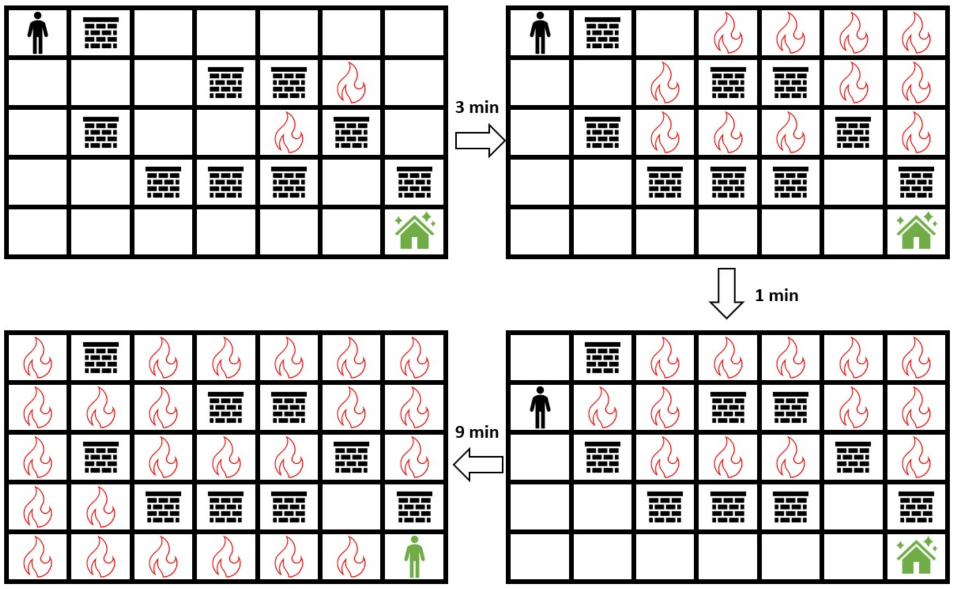
Note that even if the fire spreads to the safehouse immediately after you have reached it, it will be counted as safely reaching the safehouse.

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).

Example 1:



Input:

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

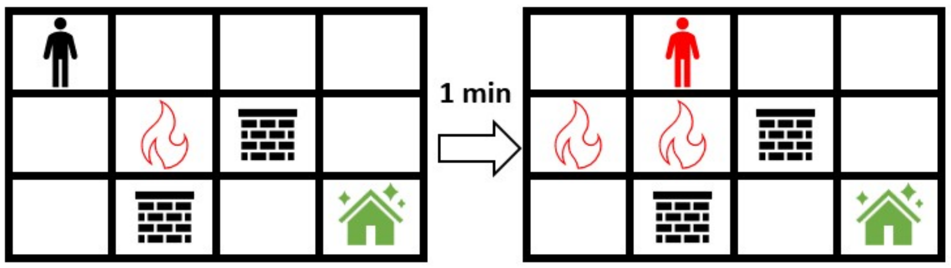
Output:

3

Explanation:

The figure above shows the scenario where you stay in the initial position for 3 minutes. You will still be able to safely reach the safehouse. Staying for more than 3 minutes will not allow you to safely reach the safehouse.

Example 2:



Input:

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

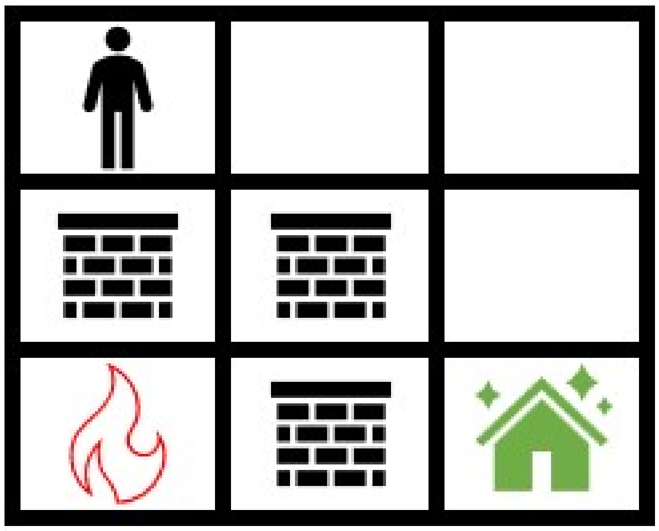
Output:

-1

Explanation:

The figure above shows the scenario where you immediately move towards the safehouse. Fire will spread to any cell you move towards and it is impossible to safely reach the safehouse. Thus, -1 is returned.

Example 3:



Input:

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

Output:

1000000000

Explanation:

The figure above shows the initial grid. Notice that the fire is contained by walls and you will always be able to safely reach the safehouse. Thus, 10

9

is returned.

Constraints:

m == grid.length

n == grid[i].length

2 <= m, n <= 300

4 <= m * n <= 2 * 10

4

grid[i][j]

is either

0

,

1

, or

2

.

grid[0][0] == grid[m - 1][n - 1] == 0

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (maximum-minutes grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Escape the Spreading Fire
* Difficulty: Hard
* 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 maximumMinutes(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Escape the Spreading Fire
* Difficulty: Hard
* 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 maximumMinutes(int[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Escape the Spreading Fire
Difficulty: Hard
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 maximumMinutes(self, grid: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Escape the Spreading Fire
* Difficulty: Hard
* 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 maximumMinutes = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Escape the Spreading Fire
* Difficulty: Hard
* 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 maximumMinutes(grid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Escape the Spreading Fire
* Difficulty: Hard
* 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 MaximumMinutes(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Escape the Spreading Fire
* Difficulty: Hard
* 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 maximumMinutes(int** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```
// Problem: Escape the Spreading Fire
// Difficulty: Hard
// 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 maximumMinutes(grid [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

Rust Solution:

```
// Problem: Escape the Spreading Fire
// Difficulty: Hard
// 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 maximum_minutes(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (maximum-minutes grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```