Problem 3286: Find a Safe Walk Through a Grid

Problem Information

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

Problem Description

You are given an

m x n

binary matrix

grid

and an integer

health

.

You start on the upper-left corner

(0, 0)

and would like to get to the lower-right corner

(m - 1, n - 1)

.

You can move up, down, left, or right from one cell to another adjacent cell as long as your health remains

positive

.

Cells

(i, j)

with

grid[i][j] = 1

are considered

unsafe

and reduce your health by 1.

Return

true

if you can reach the final cell with a health value of 1 or more, and

false

otherwise.

Example 1:

Input:

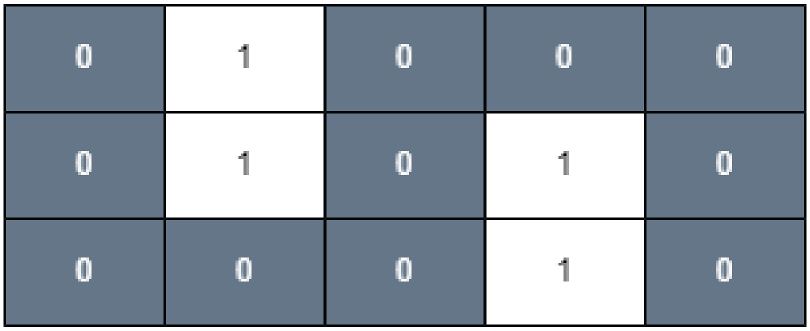
grid = [[0,1,0,0,0],[0,1,0,1,0],[0,0,0,1,0]], health = 1

Output:

true

Explanation:

The final cell can be reached safely by walking along the gray cells below.



0	1	0	0	0
0	1	0	1	0
0	0	0	1	0

Example 2:

Input:

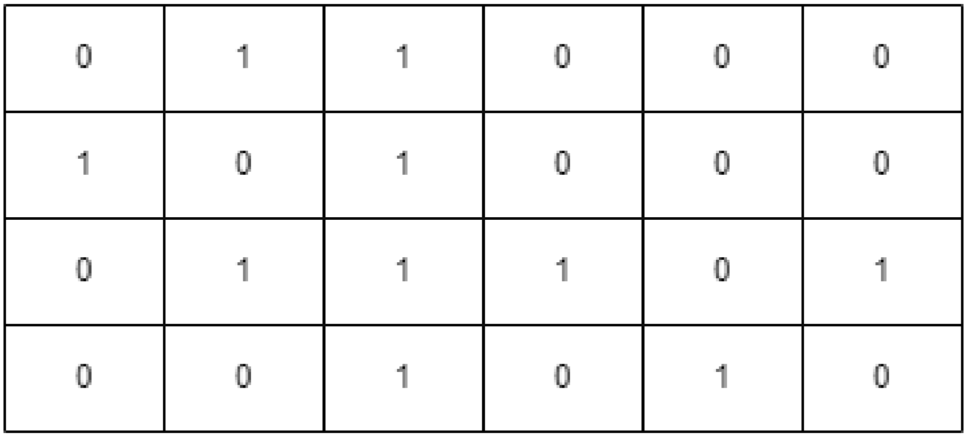
grid = [[0,1,1,0,0,0],[1,0,1,0,0,0],[0,1,1,1,0,1],[0,0,1,0,1,0]], health = 3

Output:

false

Explanation:

A minimum of 4 health points is needed to reach the final cell safely.



0	1	1	0	0	0
1	0	1	0	0	0
0	1	1	1	0	1
0	0	1	0	1	0

Example 3:

Input:

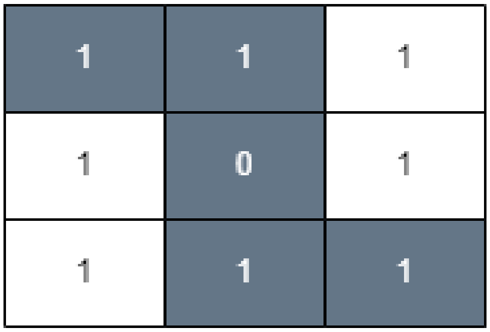
grid = [[1,1,1],[1,0,1],[1,1,1]], health = 5

Output:

true

Explanation:

The final cell can be reached safely by walking along the gray cells below.



1	1	1
1	0	1
1	1	1

Any path that does not go through the cell

(1, 1)

is unsafe since your health will drop to 0 when reaching the final cell.

Constraints:

m == grid.length

n == grid[i].length

1 <= m, n <= 50

2 <= m * n

1 <= health <= m + n

grid[i][j]

is either 0 or 1.

Code Snippets

C++:

```
class Solution {
public:
bool findSafeWalk(vector<vector<int>>& grid, int health) {

}
};
```

Java:

```
class Solution {
public boolean findSafeWalk(List<List<Integer>> grid, int health) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[][]} grid
* @param {number} health
* @return {boolean}
*/
var findSafeWalk = function(grid, health) {

};
```

TypeScript:

```
function findSafeWalk(grid: number[][], health: number): boolean {

};
```

C#:

```
public class Solution {
public bool FindSafeWalk(IList<IList<int>> grid, int health) {

}
}
```

C:

```
bool findSafeWalk(int** grid, int gridSize, int* gridColSize, int health) {

}
```

Go:

```
func findSafeWalk(grid [][]int, health int) bool {

}
```

Kotlin:

```
class Solution {
fun findSafeWalk(grid: List<List<Int>>, health: Int): Boolean {

}
}
```

Swift:

```
class Solution {
func findSafeWalk(_ grid: [[Int]], _ health: Int) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn find_safe_walk(grid: Vec<Vec<i32>>, health: i32) -> bool {

}
}
```

Ruby:

```
# @param {Integer[][]} grid
# @param {Integer} health
# @return {Boolean}
def find_safe_walk(grid, health)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $grid
* @param Integer $health
* @return Boolean
*/
function findSafeWalk($grid, $health) {

}
}
```

Dart:

```
class Solution {
bool findSafeWalk(List<List<int>> grid, int health) {

}
}
```

Scala:

```
object Solution {
def findSafeWalk(grid: List[List[Int]], health: Int): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_safe_walk(grid :: [[integer]], health :: integer) :: boolean
def find_safe_walk(grid, health) do

end
end
```

Erlang:

```
-spec find_safe_walk(Grid :: [[integer()]], Health :: integer()) ->
boolean().
find_safe_walk(Grid, Health) ->
.
```

Racket:

```
(define/contract (find-safe-walk grid health)
(-> (listof (listof exact-integer?)) exact-integer? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find a Safe Walk Through a Grid
* Difficulty: Medium
* Tags: array, graph, search, queue, heap
```

```
*
* 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:
bool findSafeWalk(vector<vector<int>>& grid, int health) {

}
};
```

Java Solution:

```
/**
* Problem: Find a Safe Walk Through a Grid
* Difficulty: Medium
* Tags: array, graph, search, queue, heap
*
* 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 boolean findSafeWalk(List<List<Integer>> grid, int health) {

}
}
```

Python3 Solution:

```
"""
Problem: Find a Safe Walk Through a Grid
Difficulty: Medium
Tags: array, graph, search, queue, heap

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 findSafeWalk(self, grid: List[List[int]], health: int) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find a Safe Walk Through a Grid
* Difficulty: Medium
* Tags: array, graph, search, queue, heap
*
* 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
* @param {number} health
* @return {boolean}
*/
var findSafeWalk = function(grid, health) {

};
```

TypeScript Solution:

```
/**
* Problem: Find a Safe Walk Through a Grid
* Difficulty: Medium
* Tags: array, graph, search, queue, heap
*
* 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 findSafeWalk(grid: number[][], health: number): boolean {

};
```

C# Solution:

```
/*
* Problem: Find a Safe Walk Through a Grid
* Difficulty: Medium
* Tags: array, graph, search, queue, heap
*
* 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 bool FindSafeWalk(IList<IList<int>> grid, int health) {

}
}
```

C Solution:

```
/*
* Problem: Find a Safe Walk Through a Grid
* Difficulty: Medium
* Tags: array, graph, search, queue, heap
*
* 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
```

```
*/

bool findSafeWalk(int** grid, int gridSize, int* gridColSize, int health) {

}
```

Go Solution:

```
// Problem: Find a Safe Walk Through a Grid
// Difficulty: Medium
// Tags: array, graph, search, queue, heap
//
// 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 findSafeWalk(grid [][]int, health int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun findSafeWalk(grid: List<List<Int>>, health: Int): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func findSafeWalk(_ grid: [[Int]], _ health: Int) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Find a Safe Walk Through a Grid
// Difficulty: Medium
// Tags: array, graph, search, queue, heap
```

```
//
// 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 find_safe_walk(grid: Vec<Vec<i32>>, health: i32) -> bool {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} grid
# @param {Integer} health
# @return {Boolean}
def find_safe_walk(grid, health)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $grid
* @param Integer $health
* @return Boolean
*/
function findSafeWalk($grid, $health) {

}
}
```

Dart Solution:

```
class Solution {
bool findSafeWalk(List<List<int>> grid, int health) {

}
}
```

Scala Solution:

```
object Solution {
def findSafeWalk(grid: List[List[Int]], health: Int): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_safe_walk(grid :: [[integer]], health :: integer) :: boolean
def find_safe_walk(grid, health) do

end
end
```

Erlang Solution:

```
-spec find_safe_walk(Grid :: [[integer()]], Health :: integer()) ->
boolean().
find_safe_walk(Grid, Health) ->
.
```

Racket Solution:

```
(define/contract (find-safe-walk grid health)
(-> (listof (listof exact-integer?)) exact-integer? boolean?)
)
```