

Problem 2812: Find the Safest Path in a Grid

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given a

0-indexed

2D matrix

grid

of size

$n \times n$

, where

(r, c)

represents:

A cell containing a thief if

$\text{grid}[r][c] = 1$

An empty cell if

$\text{grid}[r][c] = 0$

You are initially positioned at cell

$(0, 0)$

. In one move, you can move to any adjacent cell in the grid, including cells containing thieves.

The

safeness factor

of a path on the grid is defined as the

minimum

manhattan distance from any cell in the path to any thief in the grid.

Return

the

maximum safeness factor

of all paths leading to cell

$(n - 1, n - 1)$

.

An

adjacent

cell of cell

(r, c)

, is one of the cells

$(r, c + 1)$

,

(r, c - 1)

,

(r + 1, c)

and

(r - 1, c)

if it exists.

The

Manhattan distance

between two cells

(a, b)

and

(x, y)

is equal to

$$|a - x| + |b - y|$$

, where

$|val|$

denotes the absolute value of val.

Example 1:

1	0	0
0	0	0
0	0	1

Input:

```
grid = [[1,0,0],[0,0,0],[0,0,1]]
```

Output:

0

Explanation:

All paths from (0, 0) to (n - 1, n - 1) go through the thieves in cells (0, 0) and (n - 1, n - 1).

Example 2:

0	0	1
0	0	0
0	0	0

Input:

```
grid = [[0,0,1],[0,0,0],[0,0,0]]
```

Output:

2

Explanation:

The path depicted in the picture above has a safeness factor of 2 since: - The closest cell of the path to the thief at cell (0, 2) is cell (0, 0). The distance between them is $| 0 - 0 | + | 0 - 2 | = 2$. It can be shown that there are no other paths with a higher safeness factor.

Example 3:

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

Input:

```
grid = [[0,0,0,1],[0,0,0,0],[0,0,0,0],[1,0,0,0]]
```

Output:

2

Explanation:

The path depicted in the picture above has a safeness factor of 2 since: - The closest cell of the path to the thief at cell (0, 3) is cell (1, 2). The distance between them is $| 0 - 1 | + | 3 - 2 | = 2$. - The closest cell of the path to the thief at cell (3, 0) is cell (3, 2). The distance between them is $| 3 - 3 | + | 0 - 2 | = 2$. It can be shown that there are no other paths with a higher safeness factor.

Constraints:

$1 \leq \text{grid.length} == n \leq 400$

$\text{grid}[i].length == n$

$\text{grid}[i][j]$

is either

0

or

1

There is at least one thief in the

grid

Code Snippets

C++:

```
class Solution {
public:
    int maximumSafenessFactor(vector<vector<int>>& grid) {
        }
    };
}
```

Java:

```
class Solution {
public int maximumSafenessFactor(List<List<Integer>> grid) {
        }
    }
}
```

Python3:

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

Python:

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

JavaScript:

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

TypeScript:

```
function maximumSafenessFactor(grid: number[][][]): number {
}
```

C#:

```
public class Solution {
    public int MaximumSafenessFactor(IList<IList<int>> grid) {
    }
}
```

C:

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

}
```

Go:

```
func maximumSafenessFactor(grid [][]int) int {
}
```

Kotlin:

```
class Solution {  
    fun maximumSafenessFactor(grid: List<List<Int>>): Int {  
  
    }  
}
```

Swift:

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

Rust:

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

Ruby:

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

PHP:

```
class Solution {  
  
    /**  
     * @param Integer[][] $grid  
     * @return Integer  
     */  
    function maximumSafenessFactor($grid) {  
  
    }
```

```
}
```

Dart:

```
class Solution {  
    int maximumSafenessFactor(List<List<int>> grid) {  
  
    }  
}
```

Scala:

```
object Solution {  
    def maximumSafenessFactor(grid: List[List[Int]]): Int = {  
  
    }  
}
```

Elixir:

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

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
 * Problem: Find the Safest Path in 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:
    int maximumSafenessFactor(vector<vector<int>>& grid) {

    }
};
```

Java Solution:

```
/**
 * Problem: Find the Safest Path in 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 int maximumSafenessFactor(List<List<Integer>> grid) {

    }
}
```

Python3 Solution:

```
"""
Problem: Find the Safest Path in 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 maximumSafenessFactor(self, grid: List[List[int]]) -> int:
        # TODO: Implement optimized solution
        pass
```

Python Solution:

```
class Solution(object):

    def maximumSafenessFactor(self, grid):
        """
        :type grid: List[List[int]]
        :rtype: int
        """

```

JavaScript Solution:

```
/**
 * Problem: Find the Safest Path in 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
 * @return {number}
 */
var maximumSafenessFactor = function(grid) {

};
```

TypeScript Solution:

```
/**  
 * Problem: Find the Safest Path in 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 maximumSafenessFactor(grid: number[][]): number {  
  
};
```

C# Solution:

```
/*  
 * Problem: Find the Safest Path in 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 int MaximumSafenessFactor(IList<IList<int>> grid) {  
  
    }  
}
```

C Solution:

```
/*  
 * Problem: Find the Safest Path in 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
*/
int maximumSafenessFactor(int** grid, int gridSize, int* gridColSize){

}

```

Go Solution:

```

// Problem: Find the Safest Path in 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 maximumSafenessFactor(grid [][]int) int {
}

```

Kotlin Solution:

```

class Solution {
    fun maximumSafenessFactor(grid: List<List<Int>>): Int {
        }
    }

```

Swift Solution:

```

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

```

Rust Solution:

```

// Problem: Find the Safest Path in 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 maximum_safeness_factor(grid: Vec<Vec<i32>>) -> i32 {
        }

    }
}

```

Ruby Solution:

```

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

end

```

PHP Solution:

```

class Solution {

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

    }
}

```

Dart Solution:

```

class Solution {
    int maximumSafenessFactor(List<List<int>> grid) {
        }

    }
}

```

Scala Solution:

```
object Solution {  
    def maximumSafenessFactor(grid: List[List[Int]]): Int = {  
  
    }  
}
```

Elixir Solution:

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

Erlang Solution:

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

Racket Solution:

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