Problem 2850: Minimum Moves to Spread Stones Over Grid

Problem Information

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

Problem Description

You are given a

0-indexed

2D integer matrix

grid

of size

3 * 3

, representing the number of stones in each cell. The grid contains exactly

9

stones, and there can be

multiple

stones in a single cell.

In one move, you can move a single stone from its current cell to any other cell if the two cells share a side.

Return

the

minimum number of moves

required to place one stone in each cell

.

Example 1:

Input:

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

Output:

3

Explanation:

One possible sequence of moves to place one stone in each cell is: 1- Move one stone from cell (2,1) to cell (2,2). 2- Move one stone from cell (2,2) to cell (1,2). 3- Move one stone from cell (1,2) to cell (0,2). In total, it takes 3 moves to place one stone in each cell of the grid. It can be shown that 3 is the minimum number of moves required to place one stone in each cell.

Example 2:

Input:

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

Output:

4

Explanation:

One possible sequence of moves to place one stone in each cell is: 1- Move one stone from cell (0,1) to cell (0,2). 2- Move one stone from cell (0,1) to cell (1,1). 3- Move one stone from cell (2,2) to cell (1,2). 4- Move one stone from cell (2,2) to cell (2,1). In total, it takes 4 moves to place one stone in each cell of the grid. It can be shown that 4 is the minimum number of moves required to place one stone in each cell.

Constraints:

grid.length == grid[i].length == 3

0 <= grid[i][j] <= 9

Sum of

grid

is equal to

9

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (minimum-moves grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Moves to Spread Stones Over Grid
* Difficulty: Medium
* Tags: array, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Minimum Moves to Spread Stones Over Grid
* Difficulty: Medium
* Tags: array, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Moves to Spread Stones Over Grid
Difficulty: Medium
Tags: array, dp, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Moves to Spread Stones Over Grid
* Difficulty: Medium
* Tags: array, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Moves to Spread Stones Over Grid
* Difficulty: Medium
* Tags: array, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function minimumMoves(grid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Moves to Spread Stones Over Grid
* Difficulty: Medium
* Tags: array, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

C Solution:

```
/*
* Problem: Minimum Moves to Spread Stones Over Grid
* Difficulty: Medium
* Tags: array, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
```

Go Solution:

```
// Problem: Minimum Moves to Spread Stones Over Grid
// Difficulty: Medium
// Tags: array, dp, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

Rust Solution:

```
// Problem: Minimum Moves to Spread Stones Over Grid
// Difficulty: Medium
// Tags: array, dp, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (minimum-moves grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```