Problem 64: Minimum Path Sum

Problem Information

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

Problem Description

Given a

m x n

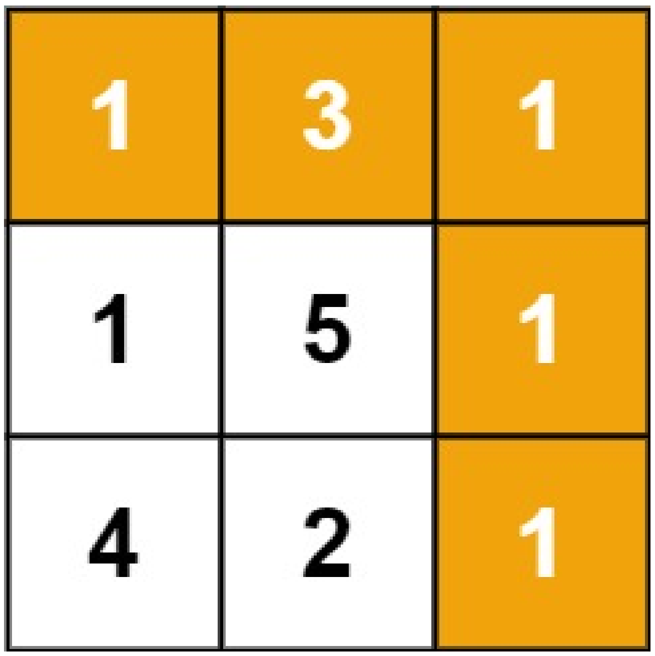
grid

filled with non-negative numbers, find a path from top left to bottom right, which minimizes the sum of all numbers along its path.

Note:

You can only move either down or right at any point in time.

Example 1:



Input:

grid = [[1,3,1],[1,5,1],[4,2,1]]

Output:

7

Explanation:

Because the path 1 → 3 → 1 → 1 → 1 minimizes the sum.

Example 2:

Input:

grid = [[1,2,3],[4,5,6]]

Output:

12

Constraints:

m == grid.length

n == grid[i].length

1 <= m, n <= 200

0 <= grid[i][j] <= 200

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}

}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (min-path-sum grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Path Sum
* Difficulty: Medium
* Tags: array, dp
*
```

```
* 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 minPathSum(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Path Sum
* Difficulty: Medium
* Tags: array, dp
*
* 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 minPathSum(int[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Path Sum
Difficulty: Medium
Tags: array, dp

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Path Sum
* Difficulty: Medium
* Tags: array, dp
*
* 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 minPathSum = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Path Sum
* Difficulty: Medium
* Tags: array, dp
```

```
*
* 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 minPathSum(grid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Path Sum
* Difficulty: Medium
* Tags: array, dp
*
* 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 MinPathSum(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Path Sum
* Difficulty: Medium
* Tags: array, dp
*
* 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 minPathSum(int** grid, int gridSize, int* gridColSize) {
}
```

Go Solution:

```
// Problem: Minimum Path Sum
// Difficulty: Medium
// Tags: array, dp
//
// 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 minPathSum(grid [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Path Sum
// Difficulty: Medium
// Tags: array, dp
//
// 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 min_path_sum(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (min-path-sum grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```