

Problem 931: Minimum Falling Path Sum

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Given an

$n \times n$

array of integers

matrix

, return

the

minimum sum

of any

falling path

through

matrix

A

falling path

starts at any element in the first row and chooses the element in the next row that is either directly below or diagonally left/right. Specifically, the next element from position

(row, col)

will be

(row + 1, col - 1)

,

(row + 1, col)

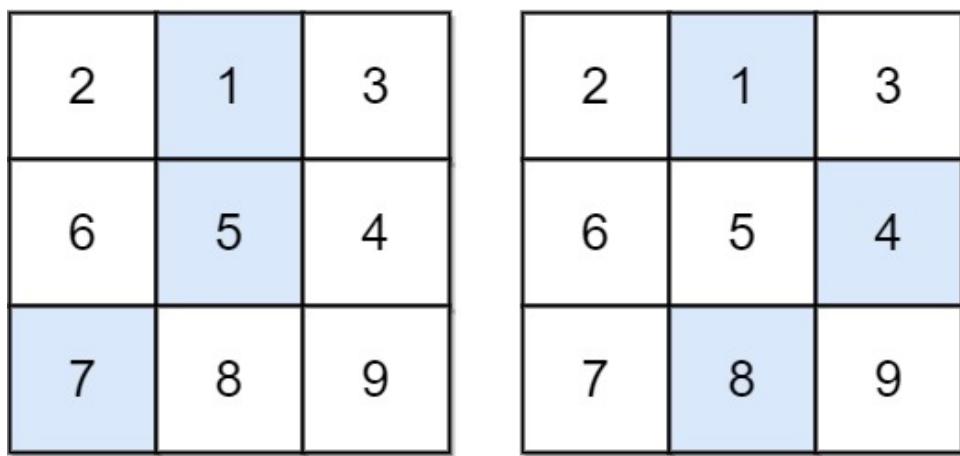
, or

(row + 1, col + 1)

.

Example 1:

2	1	3
6	5	4
7	8	9



Input:

```
matrix = [[2,1,3],[6,5,4],[7,8,9]]
```

Output:

13

Explanation:

There are two falling paths with a minimum sum as shown.

Example 2:

-19	57
-40	-5

-19	57
-40	-5

Input:

```
matrix = [[-19,57],[-40,-5]]
```

Output:

-59

Explanation:

The falling path with a minimum sum is shown.

Constraints:

```
n == matrix.length == matrix[i].length
```

```
1 <= n <= 100
```

```
-100 <= matrix[i][j] <= 100
```

Code Snippets

C++:

```
class Solution {  
public:  
    int minFallingPathSum(vector<vector<int>>& matrix) {  
  
    }  
};
```

Java:

```
class Solution {  
    public int minFallingPathSum(int[][] matrix) {  
  
    }  
}
```

Python3:

```
class Solution:  
    def minFallingPathSum(self, matrix: List[List[int]]) -> int:
```

Python:

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

JavaScript:

```
/**  
 * @param {number[][]} matrix  
 * @return {number}  
 */
```

```
var minFallingPathSum = function(matrix) {  
};
```

TypeScript:

```
function minFallingPathSum(matrix: number[][]): number {  
};
```

C#:

```
public class Solution {  
    public int MinFallingPathSum(int[][] matrix) {  
  
    }  
}
```

C:

```
int minFallingPathSum(int** matrix, int matrixSize, int* matrixColSize) {  
  
}
```

Go:

```
func minFallingPathSum(matrix [][]int) int {  
  
}
```

Kotlin:

```
class Solution {  
    fun minFallingPathSum(matrix: Array<IntArray>): Int {  
  
    }  
}
```

Swift:

```
class Solution {  
    func minFallingPathSum(_ matrix: [[Int]]) -> Int {
```

```
}
```

```
}
```

Rust:

```
impl Solution {
    pub fn min_falling_path_sum(matrix: Vec<Vec<i32>>) -> i32 {
        }
    }
```

Ruby:

```
# @param {Integer[][]} matrix
# @return {Integer}
def min_falling_path_sum(matrix)

end
```

PHP:

```
class Solution {

    /**
     * @param Integer[][] $matrix
     * @return Integer
     */
    function minFallingPathSum($matrix) {

    }
}
```

Dart:

```
class Solution {
    int minFallingPathSum(List<List<int>> matrix) {
        }
    }
```

Scala:

```

object Solution {
    def minFallingPathSum(matrix: Array[Array[Int]]): Int = {
        }
    }
}

```

Elixir:

```

defmodule Solution do
  @spec min_falling_path_sum(matrix :: [[integer]]) :: integer
  def min_falling_path_sum(matrix) do
    end
    end

```

Erlang:

```

-spec min_falling_path_sum(Matrix :: [[integer()]]) -> integer().
min_falling_path_sum(Matrix) ->
  .

```

Racket:

```

(define/contract (min-falling-path-sum matrix)
  (-> (listof (listof exact-integer?)) exact-integer?))

```

Solutions

C++ Solution:

```

/*
 * Problem: Minimum Falling 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 minFallingPathSum(vector<vector<int>>& matrix) {
        }
    };
}

```

Java Solution:

```

/**
 * Problem: Minimum Falling 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 minFallingPathSum(int[][] matrix) {

    }
}

```

Python3 Solution:

```

"""
Problem: Minimum Falling 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 minFallingPathSum(self, matrix: List[List[int]]) -> int:
        # TODO: Implement optimized solution
        pass

```

Python Solution:

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

JavaScript Solution:

```
/**
 * Problem: Minimum Falling 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[][]} matrix
 * @return {number}
 */
var minFallingPathSum = function(matrix) {

};
```

TypeScript Solution:

```
/**
 * Problem: Minimum Falling 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 minFallingPathSum(matrix: number[][]): number {
```

```
};
```

C# Solution:

```
/*
 * Problem: Minimum Falling 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 MinFallingPathSum(int[][] matrix) {
        ...
    }
}
```

C Solution:

```
/*
 * Problem: Minimum Falling 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 minFallingPathSum(int** matrix, int matrixSize, int* matrixColSize) {
    ...
}
```

Go Solution:

```
// Problem: Minimum Falling 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 minFallingPathSum(matrix [][]int) int {

}

```

Kotlin Solution:

```

class Solution {
    fun minFallingPathSum(matrix: Array<IntArray>): Int {
        return 0
    }
}

```

Swift Solution:

```

class Solution {
    func minFallingPathSum(_ matrix: [[Int]]) -> Int {
        return 0
    }
}

```

Rust Solution:

```

// Problem: Minimum Falling 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_falling_path_sum(matrix: Vec<Vec<i32>>) -> i32 {
        return 0
    }
}

```

Ruby Solution:

```
# @param {Integer[][]} matrix
# @return {Integer}
def min_falling_path_sum(matrix)

end
```

PHP Solution:

```
class Solution {

    /**
     * @param Integer[][] $matrix
     * @return Integer
     */
    function minFallingPathSum($matrix) {

    }
}
```

Dart Solution:

```
class Solution {
int minFallingPathSum(List<List<int>> matrix) {

}
```

Scala Solution:

```
object Solution {
def minFallingPathSum(matrix: Array[Array[Int]]): Int = {

}
```

Elixir Solution:

```
defmodule Solution do
@spec min_falling_path_sum(matrix :: [[integer]]) :: integer
def min_falling_path_sum(matrix) do
```

```
end  
end
```

Erlang Solution:

```
-spec min_falling_path_sum(Matrix :: [[integer()]]) -> integer().  
min_falling_path_sum(Matrix) ->  
.
```

Racket Solution:

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