Problem 1289: Minimum Falling Path Sum II

Problem Information

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

Problem Description

Given an

n x n

integer matrix

grid

, return

the minimum sum of a

falling path with non-zero shifts

.

A

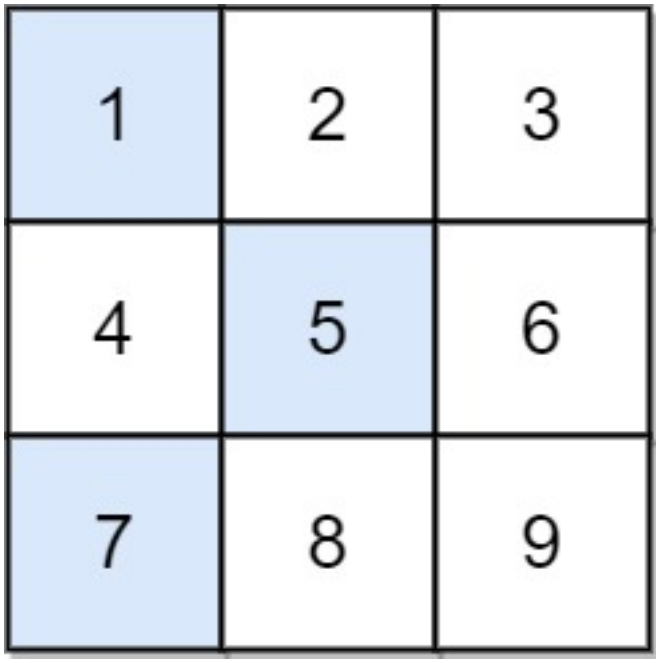
falling path with non-zero shifts

is a choice of exactly one element from each row of

grid

such that no two elements chosen in adjacent rows are in the same column.

Example 1:



Input:

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

Output:

13

Explanation:

The possible falling paths are: [1,5,9], [1,5,7], [1,6,7], [1,6,8], [2,4,8], [2,4,9], [2,6,7], [2,6,8], [3,4,8], [3,4,9], [3,5,7], [3,5,9] The falling path with the smallest sum is [1,5,7], so the answer is 13.

Example 2:

Input:

grid = [[7]]

Output:

7

Constraints:

n == grid.length == grid[i].length

1 <= n <= 200

-99 <= grid[i][j] <= 99

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

```
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Falling Path Sum II
* Difficulty: Hard
* 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>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Falling Path Sum II
* Difficulty: Hard
* 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[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Falling Path Sum II
Difficulty: Hard
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, grid: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Falling Path Sum II
* Difficulty: Hard
* 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 minFallingPathSum = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Falling Path Sum II
* Difficulty: Hard
* 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(grid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Falling Path Sum II
* Difficulty: Hard
* 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[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Falling Path Sum II
* Difficulty: Hard
* 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** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```
// Problem: Minimum Falling Path Sum II
// Difficulty: Hard
// 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(grid [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Falling Path Sum II
// Difficulty: Hard
// 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(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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