Problem 1631: Path With Minimum Effort

Problem Information

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

Problem Description

You are a hiker preparing for an upcoming hike. You are given

heights

, a 2D array of size

rows x columns

, where

heights[row][col]

represents the height of cell

(row, col)

. You are situated in the top-left cell,

(0, 0)

, and you hope to travel to the bottom-right cell,

(rows-1, columns-1)

(i.e.,

0-indexed

). You can move

up

,

down

,

left

, or

right

, and you wish to find a route that requires the minimum

effort

.

A route's

effort

is the

maximum absolute difference

in heights between two consecutive cells of the route.

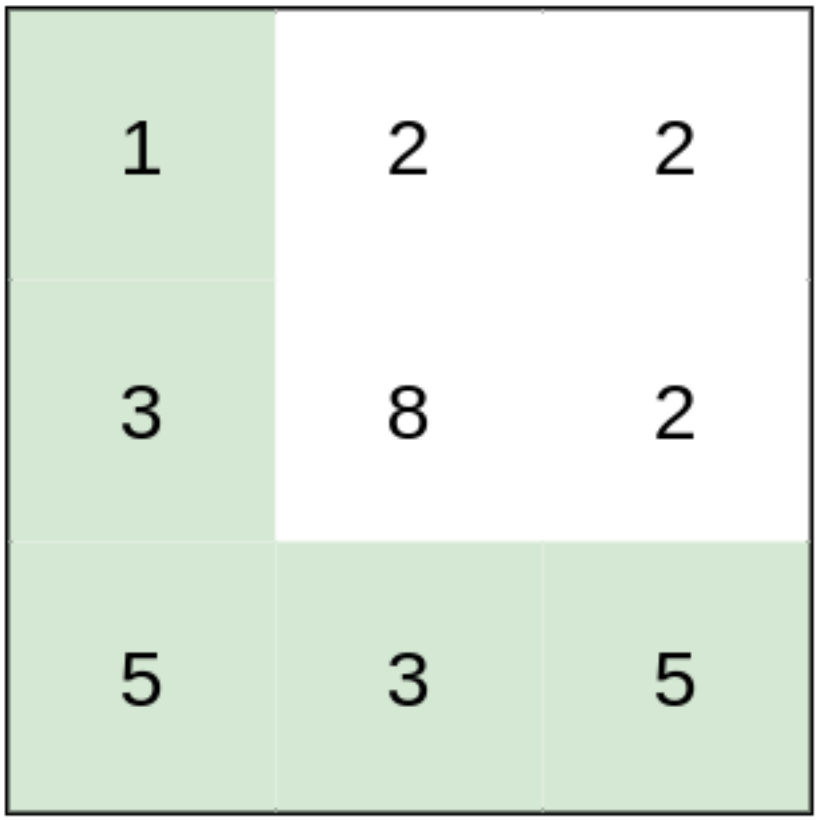
Return

the minimum

effort

required to travel from the top-left cell to the bottom-right cell.

Example 1:



Input:

heights = [[1,2,2],[3,8,2],[5,3,5]]

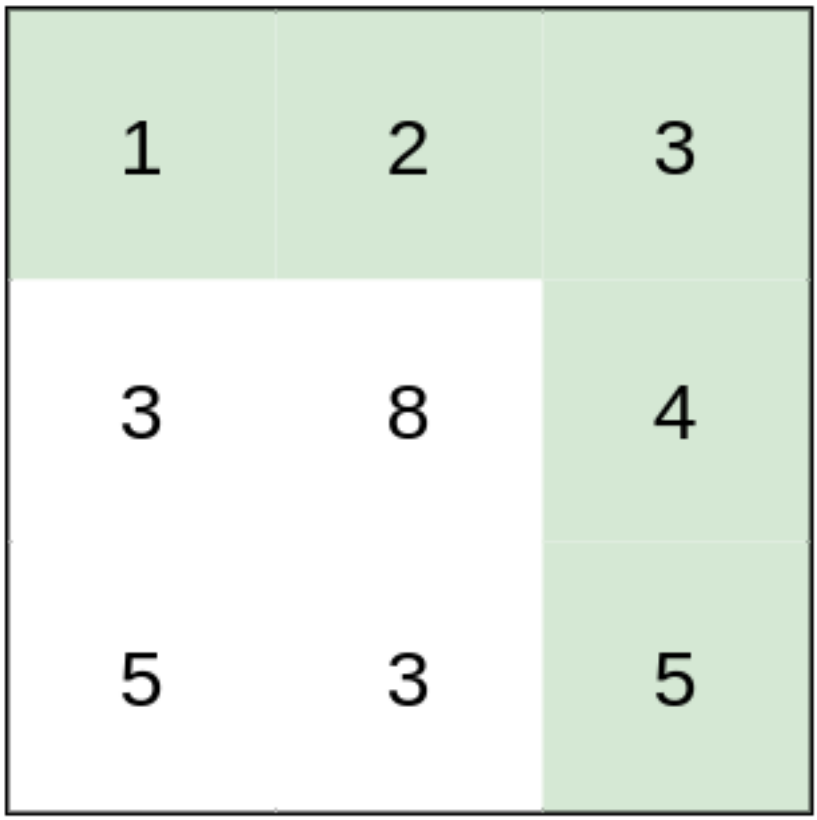
Output:

2

Explanation:

The route of [1,3,5,3,5] has a maximum absolute difference of 2 in consecutive cells. This is better than the route of [1,2,2,2,5], where the maximum absolute difference is 3.

Example 2:



Input:

heights = [[1,2,3],[3,8,4],[5,3,5]]

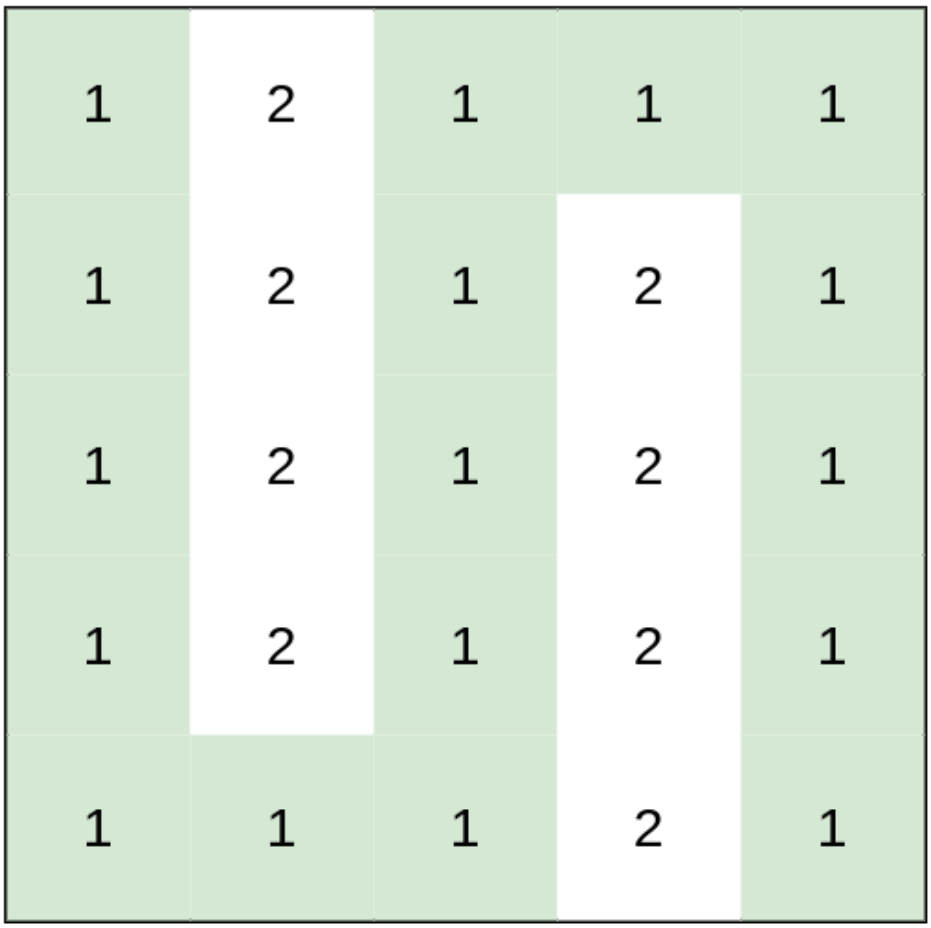
Output:

1

Explanation:

The route of [1,2,3,4,5] has a maximum absolute difference of 1 in consecutive cells, which is better than route [1,3,5,3,5].

Example 3:



Input:

heights = [[1,2,1,1,1],[1,2,1,2,1],[1,2,1,2,1],[1,2,1,2,1],[1,1,1,2,1]]

Output:

0

Explanation:

This route does not require any effort.

Constraints:

rows == heights.length

columns == heights[i].length

1 <= rows, columns <= 100

1 <= heights[i][j] <= 10

6

Code Snippets

C++:

```
class Solution {
public:
int minimumEffortPath(vector<vector<int>>& heights) {

}
};
```

Java:

```
class Solution {
public int minimumEffortPath(int[][] heights) {

}
}
```

Python3:

```
class Solution:
def minimumEffortPath(self, heights: List[List[int]]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minimumEffortPath(heights: number[][]): number {

};
```

C#:

```
public class Solution {
public int MinimumEffortPath(int[][] heights) {

}
}
```

C:

```
int minimumEffortPath(int** heights, int heightsSize, int* heightsColSize) {

}
```

Go:

```
func minimumEffortPath(heights [][]int) int {

}
```

Kotlin:

```
class Solution {
fun minimumEffortPath(heights: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func minimumEffortPath(_ heights: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_effort_path(heights: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} heights
# @return {Integer}
def minimum_effort_path(heights)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $heights
* @return Integer
*/
function minimumEffortPath($heights) {

}
}
```

Dart:

```
class Solution {
int minimumEffortPath(List<List<int>> heights) {

}
}
```

Scala:

```
object Solution {
def minimumEffortPath(heights: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_effort_path(heights :: [[integer]]) :: integer
def minimum_effort_path(heights) do

end
end
```

Erlang:

```
-spec minimum_effort_path(Heights :: [[integer()]]) -> integer().
minimum_effort_path(Heights) ->
.
```

Racket:

```
(define/contract (minimum-effort-path heights)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Path With Minimum Effort
* 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 minimumEffortPath(vector<vector<int>>& heights) {

}
};
```

Java Solution:

```
/**
* Problem: Path With Minimum Effort
* 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 minimumEffortPath(int[][] heights) {

}
}
```

Python3 Solution:

```
"""
Problem: Path With Minimum Effort
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 minimumEffortPath(self, heights: List[List[int]]) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Path With Minimum Effort
* 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[][]} heights
* @return {number}
*/
var minimumEffortPath = function(heights) {

};
```

TypeScript Solution:

```
/**
* Problem: Path With Minimum Effort
* 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 minimumEffortPath(heights: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Path With Minimum Effort
* 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 MinimumEffortPath(int[][] heights) {

}
}
```

C Solution:

```
/*
* Problem: Path With Minimum Effort
* 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 minimumEffortPath(int** heights, int heightsSize, int* heightsColSize) {

}
```

Go Solution:

```
// Problem: Path With Minimum Effort
// 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 minimumEffortPath(heights [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumEffortPath(heights: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func minimumEffortPath(_ heights: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Path With Minimum Effort
// 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 minimum_effort_path(heights: Vec<Vec<i32>>) -> i32 {

}
```

```
}
```

Ruby Solution:

```
# @param {Integer[][]} heights
# @return {Integer}
def minimum_effort_path(heights)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $heights
* @return Integer
*/
function minimumEffortPath($heights) {

}
}
```

Dart Solution:

```
class Solution {
int minimumEffortPath(List<List<int>> heights) {

}
}
```

Scala Solution:

```
object Solution {
def minimumEffortPath(heights: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_effort_path(heights :: [[integer]]) :: integer
def minimum_effort_path(heights) do

end
end
```

Erlang Solution:

```
-spec minimum_effort_path(Heights :: [[integer()]]) -> integer().
minimum_effort_path(Heights) ->
.
```

Racket Solution:

```
(define/contract (minimum-effort-path heights)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```