Problem 1368: Minimum Cost to Make at Least One Valid Path in a Grid

Problem Information

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

Problem Description

Given an

m x n

grid. Each cell of the grid has a sign pointing to the next cell you should visit if you are currently in this cell. The sign of

grid[i][j]

can be:

1

which means go to the cell to the right. (i.e go from

grid[i][j]

to

grid[i][j + 1]

)

2

which means go to the cell to the left. (i.e go from

grid[i][j]

to

grid[i][j - 1]

)

3

which means go to the lower cell. (i.e go from

grid[i][j]

to

grid[i + 1][j]

)

4

which means go to the upper cell. (i.e go from

grid[i][j]

to

grid[i - 1][j]

)

Notice that there could be some signs on the cells of the grid that point outside the grid.

You will initially start at the upper left cell

(0, 0)

. A valid path in the grid is a path that starts from the upper left cell

(0, 0)

and ends at the bottom-right cell

(m - 1, n - 1)

following the signs on the grid. The valid path does not have to be the shortest.

You can modify the sign on a cell with

cost = 1

. You can modify the sign on a cell

one time only

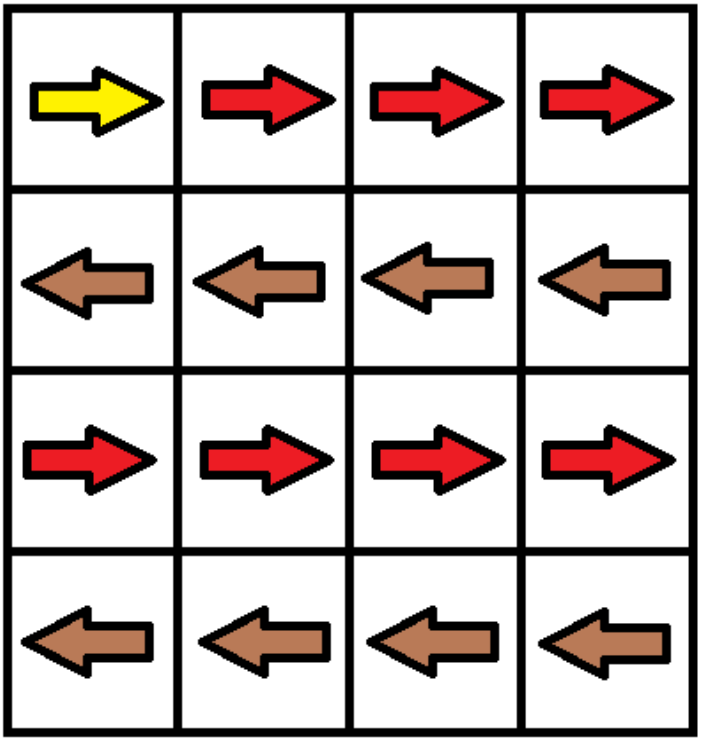
.

Return

the minimum cost to make the grid have at least one valid path

.

Example 1:



Input:

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

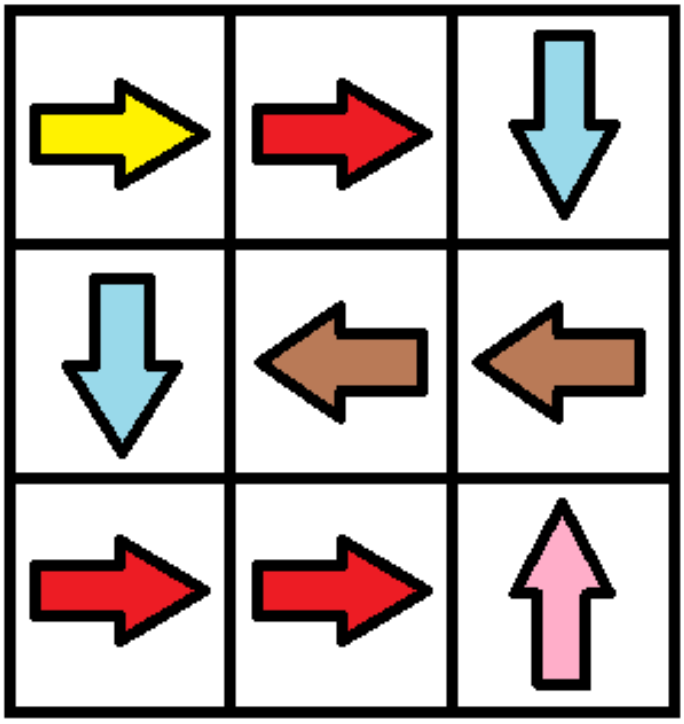
Output:

3

Explanation:

You will start at point (0, 0). The path to (3, 3) is as follows. (0, 0) --> (0, 1) --> (0, 2) --> (0, 3) change the arrow to down with cost = 1 --> (1, 3) --> (1, 2) --> (1, 1) --> (1, 0) change the arrow to down with cost = 1 --> (2, 0) --> (2, 1) --> (2, 2) --> (2, 3) change the arrow to down with cost = 1 --> (3, 3) The total cost = 3.

Example 2:



Input:

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

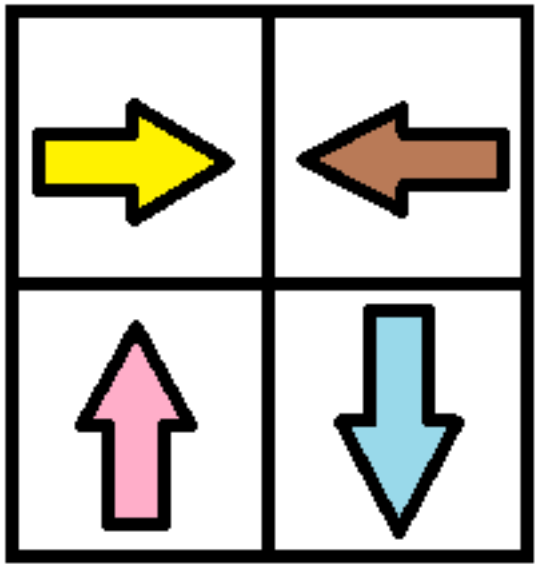
Output:

0

Explanation:

You can follow the path from (0, 0) to (2, 2).

Example 3:



Input:

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

Output:

1

Constraints:

m == grid.length

n == grid[i].length

1 <= m, n <= 100

1 <= grid[i][j] <= 4

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Cost to Make at Least One Valid Path in a Grid
* Difficulty: Hard
* 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 minCost(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Cost to Make at Least One Valid Path in a Grid
* Difficulty: Hard
* 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 minCost(int[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Cost to Make at Least One Valid Path in a Grid
Difficulty: Hard
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 minCost(self, grid: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Cost to Make at Least One Valid Path in a Grid
* Difficulty: Hard
* 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[][]} grid
* @return {number}
*/
var minCost = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Cost to Make at Least One Valid Path in a Grid
* Difficulty: Hard
* 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 minCost(grid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Cost to Make at Least One Valid Path in a Grid
* Difficulty: Hard
* 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 MinCost(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Cost to Make at Least One Valid Path in a Grid
* Difficulty: Hard
* 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 minCost(int** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```
// Problem: Minimum Cost to Make at Least One Valid Path in a Grid
// Difficulty: Hard
// 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 minCost(grid [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Cost to Make at Least One Valid Path in a Grid
// Difficulty: Hard
// 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 min_cost(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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