Problem 3596: Minimum Cost Path with Alternating Directions I

Problem Information

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

Problem Description

You are given two integers

m

and

n

representing the number of rows and columns of a grid, respectively.

The cost to enter cell

(i, j)

is defined as

(i + 1) * (j + 1)

.

The path will always begin by entering cell

(0, 0)

on move 1 and paying the entrance cost.

At each step, you move to an

adjacent

cell, following an alternating pattern:

On

odd-numbered

moves, you must move either

right

or

down

.

On

even-numbered

moves, you must move either

left

or

up

.

Return the

minimum

total cost required to reach

(m - 1, n - 1)

. If it is impossible, return -1.

Example 1:

Input:

m = 1, n = 1

Output:

1

Explanation:

You start at cell

(0, 0)

.

The cost to enter

(0, 0)

is

(0 + 1) * (0 + 1) = 1

.

Since you're at the destination, the total cost is 1.

Example 2:

Input:

m = 2, n = 1

Output:

3

Explanation:

You start at cell

(0, 0)

with cost

(0 + 1) * (0 + 1) = 1

.

Move 1 (odd): You can move down to

(1, 0)

with cost

(1 + 1) * (0 + 1) = 2

.

Thus, the total cost is

1 + 2 = 3

.

Constraints:

1 <= m, n <= 10

Code Snippets

C++:

```
class Solution {
public:
int minCost(int m, int n) {

}
};
```

Java:

```
class Solution {
public int minCost(int m, int n) {

}
}
```

Python3:

```
class Solution:
def minCost(self, m: int, n: int) -> int:
```

Python:

```
class Solution(object):
def minCost(self, m, n):
"""
:type m: int
:type n: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} m
* @param {number} n
```

```
* @return {number}
*/
var minCost = function(m, n) {

};
```

TypeScript:

```
function minCost(m: number, n: number): number {

};
```

C#:

```
public class Solution {
public int MinCost(int m, int n) {

}
}
```

C:

```
int minCost(int m, int n) {

}
```

Go:

```
func minCost(m int, n int) int {

}
```

Kotlin:

```
class Solution {
fun minCost(m: Int, n: Int): Int {

}
}
```

Swift:

```
class Solution {
func minCost(_ m: Int, _ n: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_cost(m: i32, n: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} m
# @param {Integer} n
# @return {Integer}
def min_cost(m, n)

end
```

PHP:

```
class Solution {

/**
* @param Integer $m
* @param Integer $n
* @return Integer
*/
function minCost($m, $n) {

}
}
```

Dart:

```
class Solution {
int minCost(int m, int n) {

}
```

```
}
```

Scala:

```
object Solution {
def minCost(m: Int, n: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_cost(m :: integer, n :: integer) :: integer
def min_cost(m, n) do

end
end
```

Erlang:

```
-spec min_cost(M :: integer(), N :: integer()) -> integer().
min_cost(M, N) ->
.
```

Racket:

```
(define/contract (min-cost m n)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Cost Path with Alternating Directions I
* Difficulty: Medium
* Tags: math
*
```

```
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int minCost(int m, int n) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Cost Path with Alternating Directions I
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int minCost(int m, int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Cost Path with Alternating Directions I
Difficulty: Medium
Tags: math

Approach: Optimized algorithm based on problem constraints
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""
```

```
class Solution:
def minCost(self, m: int, n: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minCost(self, m, n):
"""
:type m: int
:type n: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Cost Path with Alternating Directions I
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} m
* @param {number} n
* @return {number}
*/
var minCost = function(m, n) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Cost Path with Alternating Directions I
```

```
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

function minCost(m: number, n: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Cost Path with Alternating Directions I
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int MinCost(int m, int n) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Cost Path with Alternating Directions I
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
int minCost(int m, int n) {

}
```

Go Solution:

```
// Problem: Minimum Cost Path with Alternating Directions I
// Difficulty: Medium
// Tags: math
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

func minCost(m int, n int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minCost(m: Int, n: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minCost(_ m: Int, _ n: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Cost Path with Alternating Directions I
// Difficulty: Medium
// Tags: math
//
```

```
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn min_cost(m: i32, n: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} m
# @param {Integer} n
# @return {Integer}
def min_cost(m, n)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $m
* @param Integer $n
* @return Integer
*/
function minCost($m, $n) {

}
}
```

Dart Solution:

```
class Solution {
int minCost(int m, int n) {

}
}
```

Scala Solution:

```
object Solution {
def minCost(m: Int, n: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_cost(m :: integer, n :: integer) :: integer
def min_cost(m, n) do

end
end
```

Erlang Solution:

```
-spec min_cost(M :: integer(), N :: integer()) -> integer().
min_cost(M, N) ->
.
```

Racket Solution:

```
(define/contract (min-cost m n)
(-> exact-integer? exact-integer? exact-integer?)
)
```