Problem 3742: Maximum Path Score in a Grid

Problem Information

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

Problem Description

You are given an

m x n

grid where each cell contains one of the values 0, 1, or 2. You are also given an integer

k

.

You start from the top-left corner

(0, 0)

and want to reach the bottom-right corner

(m - 1, n - 1)

by moving only

right

or

down

.

Each cell contributes a specific score and incurs an associated cost, according to their cell values:

0: adds 0 to your score and costs 0.

1: adds 1 to your score and costs 1.

2: adds 2 to your score and costs 1.

Return the

maximum

score achievable without exceeding a total cost of

k

, or -1 if no valid path exists.

Note:

If you reach the last cell but the total cost exceeds

k

, the path is invalid.

Example 1:

Input:

grid = [[0, 1],[2, 0]], k = 1

Output:

2

Explanation:

The optimal path is:

Cell

grid[i][j]

Score

Total

Score

Cost

Total

Cost

(0, 0)

0

0

0

0

0

(1, 0)

2

2

2

1

1

(1, 1)

0

0

2

0

1

Thus, the maximum possible score is 2.

Example 2:

Input:

grid = [[0, 1],[1, 2]], k = 1

Output:

-1

Explanation:

There is no path that reaches cell

(1, 1)

without exceeding cost k. Thus, the answer is -1.

Constraints:

1 <= m, n <= 200

0 <= k <= 10

3

grid[0][0] == 0

0 <= grid[i][j] <= 2

Code Snippets

C++:

```
class Solution {
public:
int maxPathScore(vector<vector<int>>& grid, int k) {

}
};
```

Java:

```
class Solution {
public int maxPathScore(int[][] grid, int k) {

}
}
```

Python3:

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

Python:

```
class Solution(object):
def maxPathScore(self, grid, k):
```

```
"""
:type grid: List[List[int]]
:type k: int
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function maxPathScore(grid: number[][], k: number): number {

};
```

C#:

```
public class Solution {
public int MaxPathScore(int[][] grid, int k) {

}
}
```

C:

```
int maxPathScore(int** grid, int gridSize, int* gridColSize, int k) {

}
```

Go:

```
func maxPathScore(grid [][]int, k int) int {

}
```

Kotlin:

```
class Solution {
fun maxPathScore(grid: Array<IntArray>, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func maxPathScore(_ grid: [[Int]], _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_path_score(grid: Vec<Vec<i32>>, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} grid
# @param {Integer} k
# @return {Integer}
def max_path_score(grid, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $grid
* @param Integer $k
* @return Integer
*/
function maxPathScore($grid, $k) {
}
}
```

Dart:

```
class Solution {
int maxPathScore(List<List<int>> grid, int k) {

}
}
```

Scala:

```
object Solution {
def maxPathScore(grid: Array[Array[Int]], k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_path_score(grid :: [[integer]], k :: integer) :: integer
def max_path_score(grid, k) do

end
end
```

Erlang:

```
-spec max_path_score(Grid :: [[integer()]], K :: integer()) -> integer().
max_path_score(Grid, K) ->
.
```

Racket:

```
(define/contract (max-path-score grid k)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Path Score in a Grid
* 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 maxPathScore(vector<vector<int>>& grid, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Path Score in a Grid
* 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 maxPathScore(int[][] grid, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Path Score in a Grid
```

```
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 maxPathScore(self, grid: List[List[int]], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Path Score in a Grid
* 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[][]} grid
* @param {number} k
* @return {number}
*/
var maxPathScore = function(grid, k) {
};
```

TypeScript Solution:

```
/**
* Problem: Maximum Path Score in a Grid
* 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 maxPathScore(grid: number[][], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Path Score in a Grid
* 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 MaxPathScore(int[][] grid, int k) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Path Score in a Grid
* 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 maxPathScore(int** grid, int gridSize, int* gridColSize, int k) {

}
```

Go Solution:

```
// Problem: Maximum Path Score in a Grid
// 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 maxPathScore(grid [][]int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxPathScore(grid: Array<IntArray>, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxPathScore(_ grid: [[Int]], _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Path Score in a Grid
// 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 max_path_score(grid: Vec<Vec<i32>>, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} grid
# @param {Integer} k
# @return {Integer}
def max_path_score(grid, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $grid
* @param Integer $k
* @return Integer
*/
function maxPathScore($grid, $k) {

}
}
```

Dart Solution:

```
class Solution {
int maxPathScore(List<List<int>> grid, int k) {

}
}
```

Scala Solution:

```
object Solution {
def maxPathScore(grid: Array[Array[Int]], k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_path_score(grid :: [[integer]], k :: integer) :: integer
def max_path_score(grid, k) do

end
end
```

Erlang Solution:

```
-spec max_path_score(Grid :: [[integer()]], K :: integer()) -> integer().
max_path_score(Grid, K) ->
.
```

Racket Solution:

```
(define/contract (max-path-score grid k)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```