Problem 1219: Path with Maximum Gold

Problem Information

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

Problem Description

In a gold mine

grid

of size

m x n

, each cell in this mine has an integer representing the amount of gold in that cell,

0

if it is empty.

Return the maximum amount of gold you can collect under the conditions:

Every time you are located in a cell you will collect all the gold in that cell.

From your position, you can walk one step to the left, right, up, or down.

You can't visit the same cell more than once.

Never visit a cell with

0

gold.

You can start and stop collecting gold from

any

position in the grid that has some gold.

Example 1:

Input:

grid = [[0,6,0],[5,8,7],[0,9,0]]

Output:

24

Explanation:

[[0,6,0], [5,8,7], [0,9,0]] Path to get the maximum gold, 9 -> 8 -> 7.

Example 2:

Input:

grid = [[1,0,7],[2,0,6],[3,4,5],[0,3,0],[9,0,20]]

Output:

28

Explanation:

[[1,0,7], [2,0,6], [3,4,5], [0,3,0], [9,0,20]] Path to get the maximum gold, 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7.

Constraints:

m == grid.length

n == grid[i].length

1 <= m, n <= 15

0 <= grid[i][j] <= 100

There are at most

25

cells containing gold.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
```

```
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (get-maximum-gold grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Path with Maximum Gold
* Difficulty: Medium
* Tags: array
*
* 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 getMaximumGold(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Path with Maximum Gold
* Difficulty: Medium
* Tags: array
*
* 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 getMaximumGold(int[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Path with Maximum Gold
Difficulty: Medium
Tags: array
```

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Path with Maximum Gold
* Difficulty: Medium
* Tags: array
*
* 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 getMaximumGold = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Path with Maximum Gold
* Difficulty: Medium
* Tags: array
*
* 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 getMaximumGold(grid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Path with Maximum Gold
* Difficulty: Medium
* Tags: array
*
* 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 GetMaximumGold(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Path with Maximum Gold
* Difficulty: Medium
* Tags: array
*
* 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 getMaximumGold(int** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```
// Problem: Path with Maximum Gold
// Difficulty: Medium
// Tags: array
//
// 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 getMaximumGold(grid [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Path with Maximum Gold
// Difficulty: Medium
// Tags: array
```

```
//
// 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 get_maximum_gold(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (get-maximum-gold grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```