Problem 3418: Maximum Amount of Money Robot Can Earn

Problem Information

Difficulty: Medium
Acceptance Rate: 29.19%
Paid Only: No
Tags: Array, Dynamic Programming, Matrix

Problem Description

You are given an `m x n` grid. A robot starts at the top-left corner of the grid `(0, 0)` and wants to reach the bottom-right corner `(m - 1, n - 1)`. The robot can move either right or down at any point in time.

The grid contains a value `coins[i][j]` in each cell:

* If `coins[i][j] >= 0`, the robot gains that many coins. * If `coins[i][j] < 0`, the robot encounters a robber, and the robber steals the **absolute** value of `coins[i][j]` coins.

The robot has a special ability to **neutralize robbers** in at most **2 cells** on its path, preventing them from stealing coins in those cells.

Note: The robot's total coins can be negative.

Return the **maximum** profit the robot can gain on the route.

Example 1:

Input: coins = [[0,1,-1],[1,-2,3],[2,-3,4]]

Output: 8

Explanation:

An optimal path for maximum coins is:

1. Start at `(0, 0)` with `0` coins (total coins = `0`). 2. Move to `(0, 1)`, gaining `1` coin (total coins = `0 + 1 = 1`). 3. Move to `(1, 1)`, where there's a robber stealing `2` coins. The robot uses one neutralization here, avoiding the robbery (total coins = `1`). 4. Move to `(1, 2)`, gaining `3` coins (total coins = `1 + 3 = 4`). 5. Move to `(2, 2)`, gaining `4` coins (total coins = `4 + 4 = 8`).

Example 2:

Input: coins = [[10,10,10],[10,10,10]]

Output: 40

Explanation:

An optimal path for maximum coins is:

1. Start at `(0, 0)` with `10` coins (total coins = `10`). 2. Move to `(0, 1)`, gaining `10` coins (total coins = `10 + 10 = 20`). 3. Move to `(0, 2)`, gaining another `10` coins (total coins = `20 + 10 = 30`). 4. Move to `(1, 2)`, gaining the final `10` coins (total coins = `30 + 10 = 40`).

Constraints:

* `m == coins.length` * `n == coins[i].length` * `1 <= m, n <= 500` * `-1000 <= coins[i][j] <= 1000`

Code Snippets

C++:

```
class Solution {
public:
int maximumAmount(vector<vector<int>>& coins) {

}
};
```

Java:

```
class Solution {
public int maximumAmount(int[][] coins) {

}
}
```

Python3:

```
class Solution:
def maximumAmount(self, coins: List[List[int]]) -> int:
```