Problem 1463: Cherry Pickup II

Problem Information

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

Problem Description

You are given a

rows x cols

matrix

grid

representing a field of cherries where

grid[i][j]

represents the number of cherries that you can collect from the

(i, j)

cell.

You have two robots that can collect cherries for you:

Robot #1

is located at the

top-left corner

(0, 0)

, and

Robot #2

is located at the

top-right corner

(0, cols - 1)

.

Return

the maximum number of cherries collection using both robots by following the rules below

:

From a cell

(i, j)

, robots can move to cell

(i + 1, j - 1)

,

(i + 1, j)

, or

(i + 1, j + 1)

.

When any robot passes through a cell, It picks up all cherries, and the cell becomes an empty cell.

When both robots stay in the same cell, only one takes the cherries.

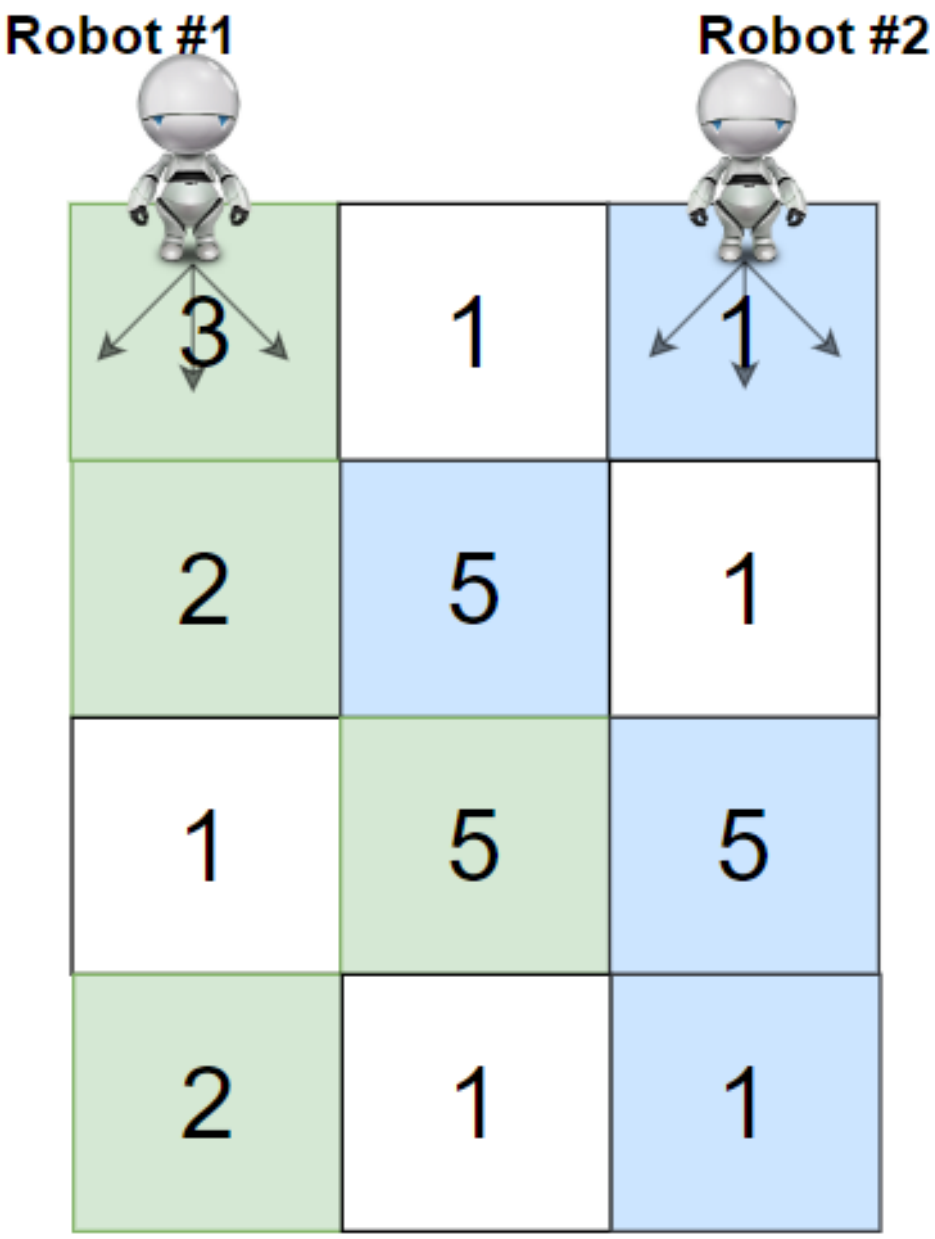
Both robots cannot move outside of the grid at any moment.

Both robots should reach the bottom row in

grid

.

Example 1:



Input:

grid = [[3,1,1],[2,5,1],[1,5,5],[2,1,1]]

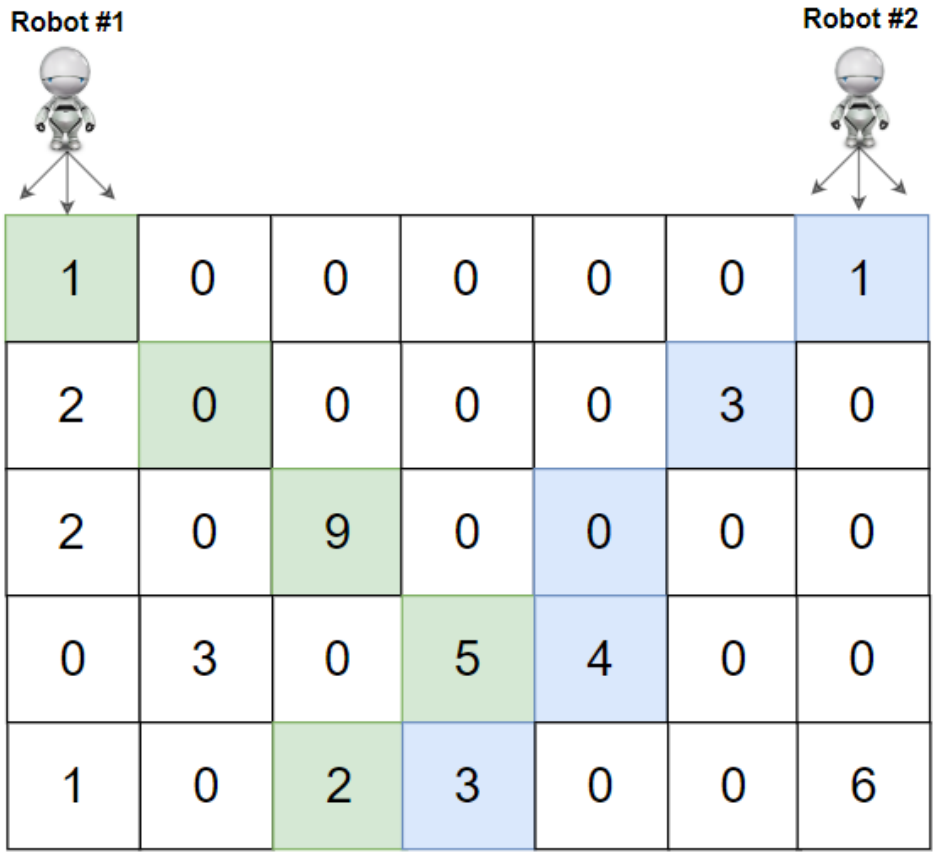
Output:

24

Explanation:

Path of robot #1 and #2 are described in color green and blue respectively. Cherries taken by Robot #1, (3 + 2 + 5 + 2) = 12. Cherries taken by Robot #2, (1 + 5 + 5 + 1) = 12. Total of cherries: 12 + 12 = 24.

Example 2:



Input:

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

Output:

28

Explanation:

Path of robot #1 and #2 are described in color green and blue respectively. Cherries taken by Robot #1, (1 + 9 + 5 + 2) = 17. Cherries taken by Robot #2, (1 + 3 + 4 + 3) = 11. Total of cherries: 17 + 11 = 28.

Constraints:

rows == grid.length

cols == grid[i].length

2 <= rows, cols <= 70

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

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

```
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (cherry-pickup grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Cherry Pickup II
* Difficulty: Hard
* 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 cherryPickup(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Cherry Pickup II
* Difficulty: Hard
* 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 cherryPickup(int[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Cherry Pickup II
Difficulty: Hard
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 cherryPickup(self, grid: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Cherry Pickup II
* Difficulty: Hard
* 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
* @return {number}
*/
var cherryPickup = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Cherry Pickup II
* Difficulty: Hard
* 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 cherryPickup(grid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Cherry Pickup II
* Difficulty: Hard
* 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 CherryPickup(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Cherry Pickup II
* Difficulty: Hard
* 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 cherryPickup(int** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```
// Problem: Cherry Pickup II
// Difficulty: Hard
// 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 cherryPickup(grid [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Cherry Pickup II
// Difficulty: Hard
// 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 cherry_pickup(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (cherry-pickup grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```