Problem 741: Cherry Pickup

Problem Information

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

Problem Description

You are given an

n x n

grid

representing a field of cherries, each cell is one of three possible integers.

0

means the cell is empty, so you can pass through,

1

means the cell contains a cherry that you can pick up and pass through, or

-1

means the cell contains a thorn that blocks your way.

Return

the maximum number of cherries you can collect by following the rules below

:

Starting at the position

(0, 0)

and reaching

(n - 1, n - 1)

by moving right or down through valid path cells (cells with value

0

or

1

).

After reaching

(n - 1, n - 1)

, returning to

(0, 0)

by moving left or up through valid path cells.

When passing through a path cell containing a cherry, you pick it up, and the cell becomes an empty cell

0

.

If there is no valid path between

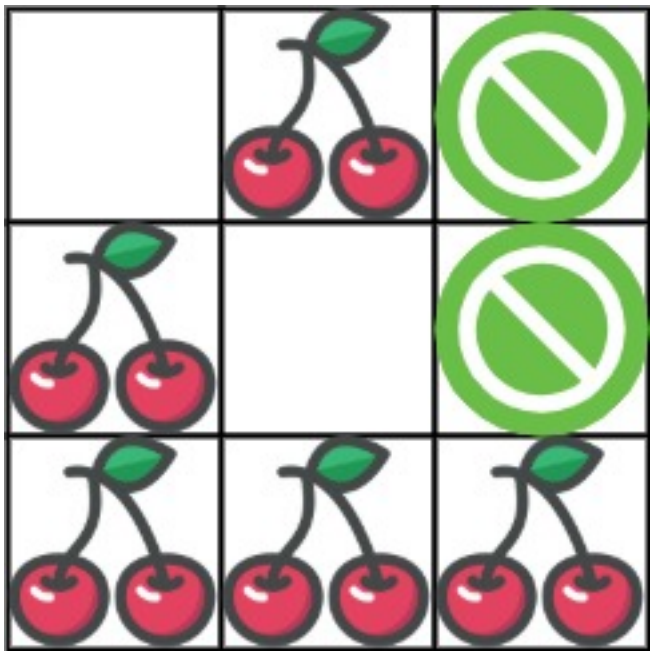
(0, 0)

and

(n - 1, n - 1)

, then no cherries can be collected.

Example 1:



Input:

grid = [[0,1,-1],[1,0,-1],[1,1,1]]

Output:

5

Explanation:

The player started at (0, 0) and went down, down, right right to reach (2, 2). 4 cherries were picked up during this single trip, and the matrix becomes [[0,1,-1],[0,0,-1],[0,0,0]]. Then, the player went left, up, up, left to return home, picking up one more cherry. The total number of cherries picked up is 5, and this is the maximum possible.

Example 2:

Input:

grid = [[1,1,-1],[1,-1,1],[-1,1,1]]

Output:

0

Constraints:

n == grid.length

n == grid[i].length

1 <= n <= 50

grid[i][j]

is

-1

,

0

, or

1

.

grid[0][0] != -1

grid[n - 1][n - 1] != -1

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
* 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
* 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
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
* 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
* 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
* 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
* 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
// 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
// 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?)
)
```