741. Cherry Pickup

DescriptionHintsSubmissionsDiscussSolution

In a 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;
- -1 means the cell contains a thorn that blocks your way.

Your task is to collect maximum number of cherries possible 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.
```

Note:

- `grid` is an `N` by `N` 2D array, with `1 <= N <= 50`.
- Each `grid[i][j]` is an integer in the set `{-1, 0, 1}`.
- It is guaranteed that grid[0][0] and grid[N-1][N-1] are not -1.

Seen this question in a real interview before?

- Difficulty:Hard
- Total Accepted:420
- Total Submissions:2.2K
- Contributor: imsure
-
- Subscribe to see which companies asked this question.

Related Topics Dynamic Programming

Similar Questions

Minimum Path Sum Dungeon Game

Approach #2: Dynamic Programming (Top Down) [Accepted]

Intuition

Instead of walking from end to beginning, let's reverse the second leg of the path, so we are only considering two paths from the beginning to the end.

Notice after `t` steps, each position `(r, c)` we could be, is on the line `r + c = t`. So if we have two people at positions `(r1, c1)` and `(r2, c2)`, then `r2 = r1 + c1 - c2`. That means the variables `r1, c1, c2` uniquely determine 2 people who have walked the same `r1 + c1` number of steps. This sets us up for dynamic programming quite nicely.

Algorithm

Let `dp[r1][c1][c2]` be the most number of cherries obtained by two people starting at `(r1, c1)` and `(r2, c2)` and walking towards `(N-1, N-1)` picking up cherries, where `r2 = r1+c1-c2`.

If `grid[r1][c1]` and `grid[r2][c2]` are not thorns, then the value of `dp[r1][c1][c2]` is `(grid[r1][c1] + grid[r2][c2])`, plus the maximum of `dp[r1+1][c1][c2]`, `dp[r1][c1+1][c2]`, `dp[r1+1][c1][c2+1]`, `dp[r1][c1+1][c2+1]` as appropriate. We should also be careful to not double count in case `(r1, c1) == (r2, c2)`.

```
//JAVA
class Solution {
    int[][][] memo;
    int[][] grid;
    int N;
    public int cherryPickup(int[][] grid) {
        this.grid = grid;
        N = grid.length;
        memo = new int[N][N][N];
        for (int[][] layer: memo)
            for (int[] row: layer)
                Arrays.fill(row, Integer.MIN_VALUE);
        return Math.max(0, dp(0, 0, 0));
    }
    public int dp(int r1, int c1, int c2) {
        int r2 = r1 + c1 - c2;
        if (N == r1 || N == r2 || N == c1 || N == c2 ||
                grid[r1][c1] == -1 || grid[r2][c2] == -1) {
            return -999999;
        } else if (r1 == N-1 && c1 == N-1) {
            return grid[r1][c1];
        } else if (memo[r1][c1][c2] != Integer.MIN_VALUE) {
            return memo[r1][c1][c2];
        } else {
            int ans = grid[r1][c1];
            if (c1 != c2) ans += grid[r2][c2];
            ans += Math.max(Math.max(dp(r1, c1+1, c2+1), dp(r1+1,
c1, c2+1)),
                          Math.max(dp(r1, c1+1, c2), dp(r1+1, c1,
c2)));
            memo[r1][c1][c2] = ans;
            return ans;
        }
    }
}
```