Maximum Number Of Moves In A Grid

solved by	Senan
Platform	LeetCode
difficulty	Medium
# Serial	2684
tags	Dynamic Programming
language	C++
solved on	@29/10/2024
link	https://leetcode.com/problems/maximum-number-of-moves-in-a-grid/
Completion	☑

Intuition

We are given a 2D grid where we can only move to neighboring cells to the right. The goal is to maximize the number of moves from any cell in the leftmost column while always moving to a strictly larger cell value. We can approach this problem using dynamic programming to keep track of the maximum moves possible from each row in each column.

Approach

1. **Initialization**: We begin with the grid dimensions, storing the number of rows `m` and columns `n`.
2. **DP Array Setup**: We use a `next` array to store the maximum moves possible starting from each row in the current column as we iterate from right to left.
3. **Backward DP Calculation**:
   - We process columns from right to left, using a `curr` array to store results for the current column based on the values from the `next` array (i.e., the next column).
   - For each cell `grid[r][c]` in the current column `c`, we check three possible moves: up-right, right, and down-right.
   - If the move is within bounds and moves to a strictly larger value, we update `curr[r]` as the maximum of its current value and `1 + next[nRow]` (representing a move to a new cell).
4. **Result Extraction**: After processing all columns, the maximum value in `next` will represent the maximum possible moves starting from any cell in the leftmost column.

Complexity

Time Complexity:

- **O(m * n)**: We iterate over all `m * n` cells once, performing constant work for each cell.

Space Complexity:

- **O(m)**: We use two arrays (`next` and `curr`) of size `m` to store intermediate results for each column.

Code

```
class Solution {
public:
    int maxMoves(vector<vector<int>>& grid) {
        int m = grid.size();
        int n = grid[0].size();
        vector<int> next(m, 0);
        for (int c = n - 2; c >= 0; c--) {
            vector<int> curr(m, 0);
            for (int r = 0; r < m; r++) {
                for (int i = -1; i <= 1; i++) {
                    int nRow = r + i;
                    if (0 <= nRow && nRow < m && grid[r][c] < grid[nRow][c + 1]) {
                        curr[r] = max(curr[r], 1 + next[nRow]);
                    }
                }
            }
            next = curr;
        }

        return *max_element(next.begin(), next.end());
    }
};
```