Given a 2D matrix matrix, handle multiple queries of the following type:

* Calculate the **sum** of the elements of matrix inside the rectangle defined by its **upper left corner** (row1, col1) and **lower right corner** (row2, col2).

Implement the NumMatrix class:

* NumMatrix(int[][] matrix) Initializes the object with the integer matrix matrix.
* int sumRegion(int row1, int col1, int row2, int col2) Returns the **sum** of the elements of matrix inside the rectangle defined by its **upper left corner** (row1, col1) and **lower right corner** (row2, col2).

**Example 1:**



**Input**

["NumMatrix", "sumRegion", "sumRegion", "sumRegion"]

[[[[3, 0, 1, 4, 2], [5, 6, 3, 2, 1], [1, 2, 0, 1, 5], [4, 1, 0, 1, 7], [1, 0, 3, 0, 5]]], [2, 1, 4, 3], [1, 1, 2, 2], [1, 2, 2, 4]]

**Output**

[null, 8, 11, 12]

**Explanation**

NumMatrix numMatrix = new NumMatrix([[3, 0, 1, 4, 2], [5, 6, 3, 2, 1], [1, 2, 0, 1, 5], [4, 1, 0, 1, 7], [1, 0, 3, 0, 5]]);

numMatrix.sumRegion(2, 1, 4, 3); // return 8 (i.e sum of the red rectangle)

numMatrix.sumRegion(1, 1, 2, 2); // return 11 (i.e sum of the green rectangle)

numMatrix.sumRegion(1, 2, 2, 4); // return 12 (i.e sum of the blue rectangle)

**Solution:**

class NumMatrix {

private int[][] dp;

public NumMatrix(int[][] matrix) {

if (matrix.length == 0 || matrix[0].length == 0) return;

dp = new int[matrix.length + 1][matrix[0].length + 1];

for (int r = 0; r < matrix.length; r++) {

for (int c = 0; c < matrix[0].length; c++) {

dp[r + 1][c + 1] = dp[r + 1][c] + dp[r][c + 1] + matrix[r][c] - dp[r][c];

}

}

}

public int sumRegion(int row1, int col1, int row2, int col2) {

return dp[row2 + 1][col2 + 1] - dp[row1][col2 + 1] - dp[row2 + 1][col1] + dp[row1][col1];

}

}

/\*\*

\* Your NumMatrix object will be instantiated and called as such:

\* NumMatrix obj = new NumMatrix(matrix);

\* int param\_1 = obj.sumRegion(row1,col1,row2,col2);

\*/

**Complexity analysis**

* Time complexity : O(1)*O*(1) time per query, O(mn)*O*(*mn*) time pre-computation. The pre-computation in the constructor takes O(mn)*O*(*mn*) time. Each sumRegion query takes O(1)*O*(1) time.
* Space complexity : O(mn)*O*(*mn*). The algorithm uses O(mn)*O*(*mn*) space to store the cumulative region sum.